

The Linux Genome: In Kernel Genetic Algorithms

Brandon Philips

September 6, 2006

Abstract

The Linux kernel manages a number of complex devices and for most of these devices a scheduler is designed to manage their use. These schedulers are like project managers managing a budget. Each project needs access to the budget but being too generous to one project will starve the others guaranteeing at least a slip in the schedule.

Much of the Linux CPU scheduler policy [1] is defined by constants that were discovered and set by Kernel developers through a process of trial and error. These constants make a reasonable compromise to perform well for most workloads but they cannot be ideal and can be improved for specific workloads.

My summer project was to investigate and implement a genetic algorithm using the in Kernel genetic library to create a self tuning scheduler.

1 The Genetic Library

In the most simple form a genetic algorithm makes a number of trials while measuring a

set of statistics. Each trial applies its genes or parameters against the algorithm that is being tuned and is ran long enough to gather useful statistics. Using these statistics the algorithm chooses the top performers, does some mutation and mixing, and reruns the algorithm.

In other words while undergoing a workload the system is constantly searching for a more optimal way to manage and schedule its resources. This can help systems running ever larger and more complex software make even better use of hardware resources.

It could be argued that instead of introducing this additional complexity into the kernel developers could simply allow system administrator to tune these constants by hand on their own system. However, this has several disadvantages: new workloads require re-tuning by the administrator, the average workload is the only workload accounted for, and it is time consuming. More than likely the administrator would be setting values based on educated guesses gathered from analyzing of benchmark results. But, what if the computer did this analysis and made educated guesses itself?

It is the perfect application for a genetic algorithm and Jake Moilanen has developed the *Genetic Library* for the Linux kernel[5] which has shown promising results:

On FFSSB there was an average improvement of 9%. On some workloads, I saw a 23% improvement. The only degradation was in sequential read, I saw a less than a 1%. Which was expected since AS is tuned for sequential read.

1.1 Terms

The idea behind genetic algorithms is pretty straight forward but there is some specific vocabulary that can get confusing:

- *gene*: a value that is a function of the measured outcome (in the case of a scheduler a *tunable*)
- *child*: a set of genes that are applied at the same time; it is a complete solution to the problem which we are searching for a solution to
- *generation*: a set of children who will be reduced through natural selection with the top performers genes being crossed over and mutated to create the next generation
- *fitness*: a value derived from a number of runtime statistics that decides the performance of a child
- *phenotype*: the end result of genes interaction; for example three different genes

work together to decide throughput of a scheduler

1.2 Lifecycle

The genetic algorithm takes a number of steps in its lifecycle. The first is running a generation and all of its children. In the I/O scheduler each child genes are applied to the scheduler and ran for 2 seconds. After the run the child's fitness is calculated from the gathered statistics. And the next child is ran.

This process repeats until all children in the generation complete. At this point the lower 50% of the generation is discarded. Then the crossover process randomly chooses genes from the surviving children to create a new generation. But, if we were to seed the new children with genes only from the most fit parents we might end up getting stuck in a local minima. So, the new children's genes are randomly mutated too.

2 The O(1) CPU Scheduler

2.1 Introduction

Choosing a process to run on the CPU is a difficult task. Desktop users expect to have highly interactive computers that can play audio, video and respond to input devices in a very quick manner.

In order to achieve interactivity a single process must not run on the processor for too long or risk the user noticing their keyboard

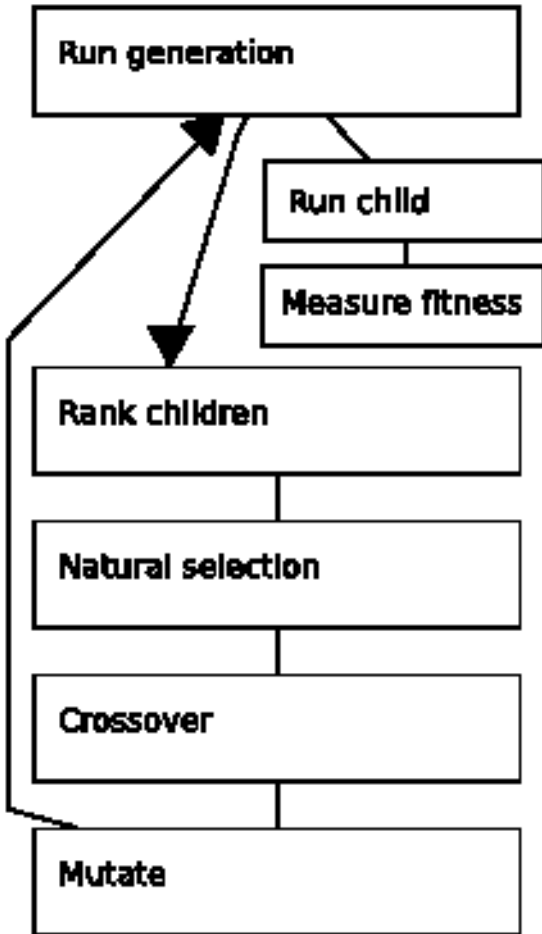


Figure 1: The lifecycle of the genetic library

presses taking a long time to appear or the audio skipping.

However, interactivity comes at a cost: context switches. When a new process is set to run on a CPU the entire state of the processor must be saved to memory so the current process can be ran at a later time. After the current process is safely saved the state of the new is copied from memory to the processor. This switching causes a great deal of memory access and is simply overhead; no useful work is being done for the user during a context switch.

So there are two competing variables that a scheduler must compromise on: *interactivity* and *efficiency*.

2.2 Timeslice

Most operating systems implement a scheme where a process has a limit on the length of time it can run before it is preempted and a new process is allowed to run. This maximum running time is called a timeslice. Under Linux the timeslice is much larger than other operating systems because a task can sleep and not lose the remainder of its timeslice. In the case of a CPU intensive process once its timeslice is up it is moved to an expired array while I/O bound processes usually remain in the active priority array (see the next section for a discussion of these data structures).

2.3 runqueues and priority arrays

The runqueue data structure is a per processor data structure that tracks all active and expired processes on a CPU:

```
struct rq { spinlock_t lock;
unsigned long nr_running; /* runnable
task count */
unsigned long long nr_switches; /*
number of context switches */
struct task_struct *curr; /*
currently running tasks */
struct prio_array *active; /* active
bitmap and queue */
struct prio_array *expired; /*
expired bitmap and queue */
struct prio_array arrays[2]; /*
pointers to the two queues */
}
```

The Linux CPU scheduler is considered to be an $O(1)$ algorithm thanks to the priority arrays (Figure 2). A 1 in active priority array bitmap means that there is a process in the runqueue that is ready to be run at this priority.

So why $O(1)$? Because a single machine instruction can be used to find the first 1 in; the index of this bit is added to a pointer into a round robin queue of all tasks at that dynamic priority level. So, with a constant number of instructions the Kernel knows the next process to run.

The expired array slowly fills with processes that aren't interactive enough to be immediately reinserted into the runqueue. The scheduler keeps track of the number of processes in this array and the amount of time

that they have been waiting to run. When this expiration time is reached the remaining tasks in the active array are moved to the expired array and the pointers are swapped. It isn't constant time, but moving pointers on a small number of tasks is quite fast.

2.4 Linux Scheduler Tunables

These are the tuning knobs that can be used to modify the behaviour of the Kernel's scheduler at compile time.

- `MIN_TIMESLICE` (5msecs): The smallest timeslice a process can receive.
- `DEF_TIMESLICE` (100msecs): The default timeslice for a process with a nice value of 0. Timeslice scaling happens for all other nice values.
- `ON_RUNQUEUE_WEIGHT` (30):
- `CHILD_PENALTY` (95): Percentage of the parents `sleep_avg` that a child receives after forking.
- `PARENT_PENALTY` (100): Percentage of the parents `sleep_avg` that a parent keeps after forking a child.
- `EXIT_WEIGHT` (3): The parent of an exiting CPU hog gets a `child_sleep_avg/EXIT_WEIGHT` penalty.
- `PRIO_BONUS_RATIO` (25): Percentage of the dynamic priority range that a task can achieve. In other words it defines how well the static user defined nice values are honored.

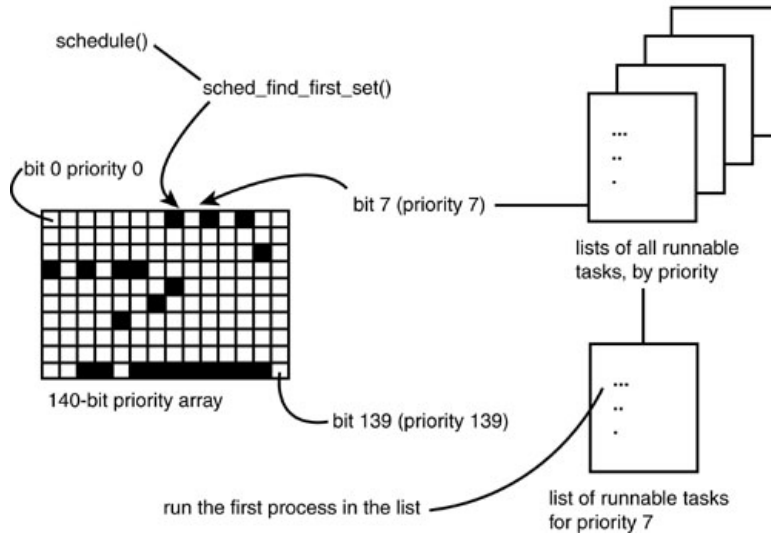


Figure 2: Replace text here with your desired caption.

- `INTERACTIVE_DELTA` (2): An interactive process is reinserted into the run-queue after its timeslice has expired. How interactive a task must be in order to be reinserted is decided by its nice value. This limit is scaled linearly and offset by this delta.

3 Scheduler Statistics

Choosing the statistics that are going to be used by a genetic algorithm is an extremely important task. The statistics that are going to be used as input into choosing genes must have a few characteristics:

- Genes affect the value: A statistic that isn't a function of the value of the gene who's fate it is deciding is not useful.
- Users affect the value: A statistic must

also be a function of the users action otherwise there would be no need for a genetic algorithm.

- Statistics values must unaffected by past genes: When a statistic is captured its value should be as independent of past gene values as possible.

3.1 I/O Scheduler

The Genetic Anticipatory I/O Scheduler patch uses five statistics to tune the genes:

- reads: number of read requests
- writes: number of write requests
- read_sectors: total number of sectors requested for read
- write_sectors: total number of sectors requested for write

- `time_in_queue`: the total amount of time requests have been waiting in the queue

3.2 CPU Scheduler

The Genetic CPU Scheduler uses three statistics to tune its genes:

- `cpu_time`: total amount of time processes spent on the CPU
- `run_delay`: the amount of time it took the average process to move from runnable to running
- `pcnt`: the number of context switches

4 Genetic CPU Benchmark Results

At the time of writing benchmarks under Kernbench, SPECJBB, and hackbench were inconclusive for the CPU scheduler. More investigation and work will need to be done.

5 genetk

One of the tools that was needed this summer was a realtime visualization of the genetic algorithm. I used the TCL/TK language and GUI framework to develop this tool. I choose TCL/TK because I had never developed in it and because it is one of the few GUI frameworks that is guaranteed to run almost anywhere.

An example of one of the graphs can be seen in Figure 3.

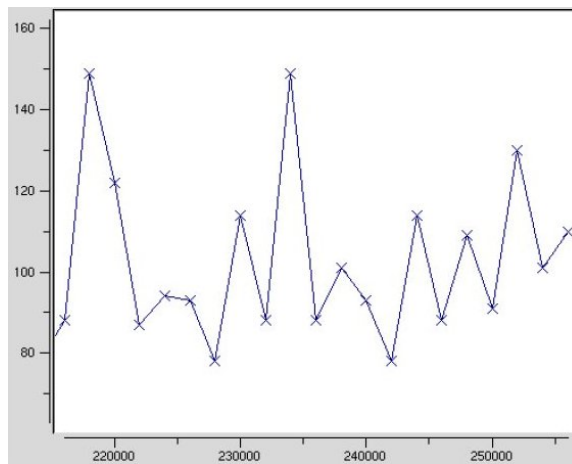


Figure 3: Real time graphing of the values that the genetic library has been trying for the `write_batch_expire` gene of the anticipatory I/O scheduler.

6 Linux Kernel Development

Linux Kernel Development revolves around the Linux Kernel Mailing list. This email list has traffic peaking at around 300 hundred messages a day. Topics on this list generally include bug reports, patches, code review and feature requests. But of all of these topics patches are the real star- they are the lifeblood of the Linux development process.

6.1 Submitting Patches

Attached to this document are all of the genetic library patches that I have been tweaking or developing over the course of the summer.

These patches are formatted in a very spe-

cific manner and should be thought of as individual proofs or theorems that build on top of each other. The primary rule in submitting a patch is to make it easy on the subsystem maintainers.

For this reason each patch should be thought of as an individual mathematical proof that contains these elements:

- Subject: A descriptive subject for your patch including a [PATCH] prefix
- A descriptive body that defends the need of this patch and its purpose
- A “Signed-off-by: Developer jdev@example.comj” line that certifies you are the author of this code
- The patch in ”diff -uprN“ format with a patch level of one

In the case of the genetic library there are several patches that make up the entire *patchset*. After applying each patch the Kernel is left in a compilable and runnable state which is important in case one of your patches is dropped and requires work later in the development process. Separating out patches in this manner also forces the developer to think about why and where a certain change should be made.

6.2 Code Review

Reviewing patches is hard work but dozens of messages a day pick apart proposed patches on the LKML mailing lists. Because patches

are sent as regular ASCII emails patch critiquing happens using the usual reply function in email clients and specific pieces of code are quoted and referenced. The system works quite well and developers can learn from this meticulous process.

Unfortunately I was unable to submit the Genetic CPU Scheduler patch sets to LKML this summer but Jake provided me a taste of this code review. Attached to the end of this document is the code review of my patches that was done half way through the summer.

7 Development Tools

A good developer knows how to make use of tools to automate complex, boring and error-prone processes. The Linux Kernel community has a great set of tools available to it and these are a few that I found most useful during my work.

7.1 ketchup

While developing kernel patches it is important to stay on top of the latest kernel development. This is important when you send patches to the Linux Kernel Mailing List for review because developers want the patches to apply cleanly on top of the most recent kernel tree.

In general patches should be based on top of the latest release candidate. These releases are made every few weeks during the development phase of a kernel release.

In order to stay up to date with these releases ketchup [7] can be used:

```
$ ketchup 2.6-rc
```

With that simple command the latest 2.6 series Linux Kernel is checked out into the current directory ready for hacking and building.

7.2 quilt

Kernel development is based on the flow of patches but running `diff -uprN` against two kernel trees to generate patches can be a real pain. Furthermore keeping track of which patches are applied to your tree and the order in which they have been applied can be a real pain.

Luckily `quilt`[8] was developed to make working with patches easy. This tool is based on the idea of a stack of patches. So, patches are applied and stacked on top of the tree. Working with `quilt` is very easy:

```
1. $ quilt import
genetic-lib.patch
io-fingerprinting.patch
genetic-io-sched.patch
genetic-as-sched.patch
2. $ quilt push -a
3. $ quilt new
genetic-cpu-sched.patch
4. $ quilt edit kernel/sched.c
...hack hack hack ...
5. $ quilt refresh
6. $ quilt pop -a
```

In this example I am starting my work on the genetic library. In the first step I import all of the original genetic library patches. Then the patches are pushed on the tree in the order in which they were imported so that patches that build on top (see Figure 5) ap-

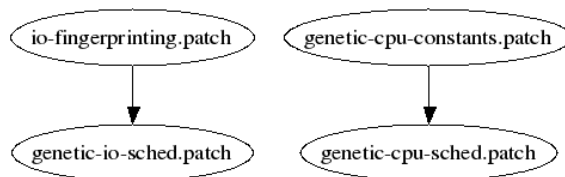


Figure 4: Patches can have dependencies when they change the same files

ply. Step three creates a new patches that is put on top of the stack. Then a file is added to the patch and edited. After the developer has done his work a refresh does the diff between the original and edited file to add the changes to the patch. The final step pops the changes off the tree reverting the patches in order.

7.3 autotest

The genetic library is about performance so benchmarks are important. However, a really easy to use and scriptable testing harness was not available that supported all of the benchmarks I wanted to run. So, I wrote some patches for the autotest project.

This project spear headed by Martin Bligh is written in Python and has the goal of becoming a standard harness for testing the Kernel.

The usage is fairly straight forward:

```
cd /usr/local svn checkout
svn://test.kernel.org/autotest/trunk
autotest cd /usr/local/autotest
bin/autotest control_file
```

Autotest will compile all tests in the control file, run profilers during the run, and capture the test output. And since the con-

trol files are written in Python doing complex scripting of the runs of tests is straightforward.

7.4 qemu

QEMU is the perfect tool for developing core kernel code like schedulers and file systems. Like the commercially available program VMWare QEMU does full system emulation. Although this emulation is slow compared to running the software natively it has several advantages:

- Fast reboots: If a QEMU machine crashes simply kill the process, fix the bug and try again.
- Contained damage: Any coding errors that could cause damage to your system only destroys the virtual hard disk- your real data is safe.
- Easy capture of oopses: When a Kernel crashes under QEMU capturing back traces is as simple as copying text from a X terminal.

Mounting a QEMU disk image, copying over the Kernel image, and configuring the boot loader can be real pain. Luckily QEMU simplifies this process by exposing a flag to bootstrap a Kernel from the host system:

```
qemu -kernel arch/i386/boot/bzImage
-append root=/dev/hda1 disk.img
```

With that command a QEMU machine will boot a freshly compiled directly from a kernel tree.

8 Conclusion

Working with and learning from Jake Moilanen and the rest of the IBM Austin LTC team has been a great experience. Particularly I appreciate the freedom I was given to investigate new tools like TCL/TK and autotest to solve problems on this project. And although the benchmark results are inconclusive up to this point I have two more weeks to try to improve this situation.

References

- [1] Josh Aas, *Linux 2.6.8.1 CPU Scheduler*. Silicon Graphics Inc, <http://josh.trancesoftware.com/linux/2005>.
- [2] Craig Thomas, *Linux Process Scheduler Improvements in Version 2.6.0*. Open Source Development Labs, <http://developer.osdl.org/craiger/hackbench/2003>.
- [3] M. Time Jones, *Inside the Linux scheduler*. Emulex, <http://www-128.ibm.com/developerworks/linux/library/l-scheduler/index.html> 2006.
- [4] Robert Love, *The Linux Process Scheduler*. Sams Publishing, <http://samspublishing.com/articles/article.asp?p=1017> 2003.
- [5] Jake Moilanen, *Using genetic algorithms to autonomically tune the kernel*, IBM, <http://www.linuxsymposium.org/2005/linuxsymposium2005>.

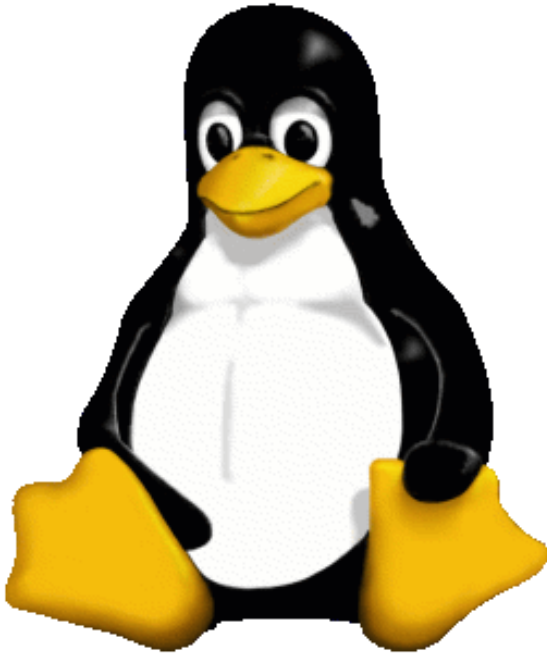


Figure 5: And don't forget to hug a penguin!

- [6] Jake Moilanen, *I/O Workload Fingerprinting*, IBM, http://www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf, 2005.
- [7] Matt Mackall, *Ketchup*, <http://www.selenic.com/ketchup/wiki/>
- [8] <http://savannah.nongnu.org/projects/quilt>

Sep 05, 06 23:45

001-export-cpu-scheduler-email

Page 1/3

A patch I sent to LKML to ease kernel scheduler development and testing.

```
From philips@ifup.org Sun Sep  3 08:28:36 2006
Date: Sun, 3 Sep 2006 08:28:02 -0500
From: Brandon Philips <brandon@ifup.org>
To: mingo@elte.hu, rml@tech9.net
Cc: "Brandon D. Philips" <brandon@ifup.org>,
    linux-kernel@vger.kernel.org
Subject: [PATCH] Export CPU Scheduler Tunables via DebugFS
```

This patch exports the CPU scheduler tunables via DebugFS.

```
philips@plankton:~$ ls /debug/cpu-scheduler/
child_penalty  interactive_delta  min_timeslice      prio_bonus_ratio
def_timeslice  max_bonus          on_runqueue_weight  starvation_limit
exit_weight    max_sleep_avg      parent_penalty
```

It is similar to the patches by Robert Love and Ingo Molnar that were available for the 2.5 series: <http://kerneltrap.org/node/525/1938>

The goal remains the same: offer a simple way for `_developers_` to tune and debug the scheduler.

Perhaps someone could even automate the testing of different values[1] under different workloads[2].

[1] <http://meikon.homeip.net/extras/papers/linux-2.5-scheduler-analysis.pdf>
 [2] <http://test.kernel.org/autotest/QuickStart>

Signed-off-by: Brandon D. Philips <brandon@ifup.org>

```
---
kernel/sched.c      | 115 +++++++++++++++++++++++++++++++++++++++++++++++++++++
lib/Kconfig.debug  |   8 +++
2 files changed, 112 insertions(+), 11 deletions(-)
```

Index: linux-rc/kernel/sched.c

```
=====
--- linux-rc.orig/kernel/sched.c
+++ linux-rc/kernel/sched.c
@@ -52,6 +52,7 @@
#include <linux/acct.h>
#include <linux/kprobes.h>
#include <linux/delayacct.h>
+#include <linux/debugfs.h>
#include <asm/tlb.h>

#include <asm/unistd.h>
@@ -87,17 +88,60 @@
 * default timeslice is 100 msecs, maximum timeslice is 800 msecs.
 * Timeslices get refilled after they expire.
 */
-#define MIN_TIMESLICE          max(5 * HZ / 1000, 1)
-#define DEF_TIMESLICE          (100 * HZ / 1000)
-#define ON_RUNQUEUE_WEIGHT    30
-#define CHILD_PENALTY         95
-#define PARENT_PENALTY        100
-#define EXIT_WEIGHT            3
-#define PRIO_BONUS_RATIO      25
-#define MAX_BONUS              (MAX_USER_PRIO * PRIO_BONUS_RATIO / 100)
-#define INTERACTIVE_DELTA      2
-#define MAX_SLEEP_AVG          (DEF_TIMESLICE * MAX_BONUS)
-#define STARVATION_LIMIT      (MAX_SLEEP_AVG)
+#define DEFAULT_MIN_TIMESLICE  max(5 * HZ / 1000, 1)
+#define DEFAULT_DEF_TIMESLICE  (100 * HZ / 1000)
+#define DEFAULT_ON_RUNQUEUE_WEIGHT  30
+#define DEFAULT_CHILD_PENALTY     95
+#define DEFAULT_PARENT_PENALTY    100
+#define DEFAULT_EXIT_WEIGHT        3
+#define DEFAULT_PRIO_BONUS_RATIO  25
+#define DEFAULT_MAX_BONUS          (MAX_USER_PRIO * DEFAULT_PRIO_BONUS_RATIO / 100)
+#define DEFAULT_INTERACTIVE_DELTA  2
+#define DEFAULT_MAX_SLEEP_AVG      (DEFAULT_DEF_TIMESLICE * DEFAULT_MAX_BONUS)
+#define DEFAULT_STARVATION_LIMIT  (DEFAULT_MAX_SLEEP_AVG)
+#define DEFAULT_NS_MAX_SLEEP_AVG  (JIFFIES_TO_NS(DEFAULT_MAX_SLEEP_AVG))
+
+
+/*
+ * Simply use the static definitions if we are not exporting tunables via
+ * DebugFS for runtime tuning.
+ */
+#ifndef CONFIG_DEBUGFS_SCHED
+#define MIN_TIMESLICE          DEFAULT_MIN_TIMESLICE
+#define DEF_TIMESLICE          DEFAULT_DEF_TIMESLICE
+#define ON_RUNQUEUE_WEIGHT    DEFAULT_ON_RUNQUEUE_WEIGHT
+#define CHILD_PENALTY         DEFAULT_CHILD_PENALTY
+#define PARENT_PENALTY        DEFAULT_PARENT_PENALTY
+#define EXIT_WEIGHT            DEFAULT_EXIT_WEIGHT
+#define PRIO_BONUS_RATIO      DEFAULT_PRIO_BONUS_RATIO
+#define MAX_BONUS              DEFAULT_MAX_BONUS
+#define INTERACTIVE_DELTA      DEFAULT_INTERACTIVE_DELTA
+#define MAX_SLEEP_AVG          DEFAULT_MAX_SLEEP_AVG
+#define STARVATION_LIMIT      DEFAULT_STARVATION_LIMIT
+#else
+int min_timeslice;
+#define MIN_TIMESLICE          (min_timeslice)
+int def_timeslice;
+#define DEF_TIMESLICE          (def_timeslice)
+int on_runqueue_weight;
+#define ON_RUNQUEUE_WEIGHT    (on_runqueue_weight)
+int child_penalty;
+#define CHILD_PENALTY         (child_penalty)
```

Tuesday September 05, 2006

1/4

```

+int parent_penalty;
+#define PARENT_PENALTY      (parent_penalty)
+int exit_weight;
+#define EXIT_WEIGHT        (exit_weight)
+int prio_bonus_ratio;
+#define PRIO_BONUS_RATIO   (prio_bonus_ratio)
+int max_bonus;
+#define MAX_BONUS          (max_bonus)
+int interactive_delta;
+#define INTERACTIVE_DELTA  (interactive_delta)
+int max_sleep_avg;
+#define MAX_SLEEP_AVG      (max_sleep_avg)
+int starvation_limit;
+#define STARVATION_LIMIT   (starvation_limit)
+#endif
+
+#define NS_MAX_SLEEP_AVG    (JIFFIES_TO_NS(MAX_SLEEP_AVG))

/*
@@ -6725,10 +6769,59 @@ int in_sched_functions(unsigned long addr
    && addr < (unsigned long)__sched_text_end);
}

+#ifdef CONFIG_DEBUGFS_SCHED
+
+#define debugfs_sched_create(name, parent) \
+debugfs_create_u32(#name, 0644, parent, &name)
+
+int __init init_debugfs(void)
+{
+    struct dentry * root;
+
+    if ((root = debugfs_create_dir("cpu-scheduler", NULL)) == NULL)
+        goto err_root;
+
+    debugfs_sched_create(min_timeslice, root);
+    debugfs_sched_create(def_timeslice, root);
+    debugfs_sched_create(on_runqueue_weight, root);
+    debugfs_sched_create(child_penalty, root);
+    debugfs_sched_create(parent_penalty, root);
+    debugfs_sched_create(exit_weight, root);
+    debugfs_sched_create(prio_bonus_ratio, root);
+    debugfs_sched_create(max_bonus, root);
+    debugfs_sched_create(interactive_delta, root);
+    debugfs_sched_create(max_sleep_avg, root);
+    debugfs_sched_create(starvation_limit, root);
+
+err_root:
+    return 0;
+}
+postcore_initcall(init_debugfs);
+
+void __init init_tunables(void)
+{
+    min_timeslice =      DEFAULT_MIN_TIMESLICE;
+    def_timeslice =     DEFAULT_DEF_TIMESLICE;
+    on_runqueue_weight = DEFAULT_ON_RUNQUEUE_WEIGHT;
+    child_penalty =     DEFAULT_CHILD_PENALTY;
+    parent_penalty =    DEFAULT_PARENT_PENALTY;
+    exit_weight =       DEFAULT_EXIT_WEIGHT;
+    prio_bonus_ratio =  DEFAULT_PRIO_BONUS_RATIO;
+    max_bonus =         DEFAULT_MAX_BONUS;
+    interactive_delta = DEFAULT_INTERACTIVE_DELTA;
+    max_sleep_avg =     DEFAULT_MAX_SLEEP_AVG;
+    starvation_limit =  DEFAULT_STARVATION_LIMIT;
+}
+#endif
+
+void __init sched_init(void)
+{
+    int i, j, k;

+#ifdef CONFIG_DEBUGFS_SCHED
+    init_tunables();
+#endif
+
+    for_each_possible_cpu(i) {
+        struct prio_array *array;
+        struct rq *rq;
Index: linux-rc/lib/Kconfig.debug
=====
--- linux-rc.orig/lib/Kconfig.debug
+++ linux-rc/lib/Kconfig.debug
@@ -93,6 +93,14 @@ config SCHEDSTATS
     application, you can say N to avoid the very slight overhead
     this adds.

+config DEBUGFS_SCHED
+    bool "Export CPU Scheduler Tunables"
+    depends on DEBUG_KERNEL && DEBUG_FS
+    help
+        If you say Y here, CPU Scheduler tuning knobs will be available via
+        debugfs in /debug/cpu-scheduler/. Note: No sanity checking is done
+        on the values.
+
 config DEBUG_SLAB
     bool "Debug slab memory allocations"
     depends on DEBUG_KERNEL && SLAB
--

```


Sep 04, 06 17:14

002-export-cpu-scheduler-email

Page 1/1

From linux-kernel-owner+brandon=40ifup.org-S1751435AbWICRMG@vger.kernel.org Sun Sep 3 12:13:22 2006
 Date: Sun, 3 Sep 2006 19:12:02 +0200
 From: Andreas Mohr <andi@rhlx01.fht-esslingen.de>
 To: Brandon Philips <brandon@ifup.org>
 Cc: mingo@elte.hu, rml@tech9.net, linux-kernel@vger.kernel.org
 Subject: Re: [PATCH] Export CPU Scheduler Tunables via DebugFS

Hi,

On Sun, Sep 03, 2006 at 08:28:02AM -0500, Brandon Philips wrote:
 > This patch exports the CPU scheduler tunables via DebugFS.

```
> +int min_timeslice;
> +#define MIN_TIMESLICE          (min_timeslice)
> +int def_timeslice;
> +#define DEF_TIMESLICE          (def_timeslice)
> +int on_runqueue_weight;
> +#define ON_RUNQUEUE_WEIGHT    (on_runqueue_weight)
> +int child_penalty;
> +#define CHILD_PENALTY          (child_penalty)
> +int parent_penalty;
> +#define PARENT_PENALTY        (parent_penalty)
> +int exit_weight;
> +#define EXIT_WEIGHT            (exit_weight)
> +int prio_bonus_ratio;
> +#define PRIO_BONUS_RATIO       (prio_bonus_ratio)
> +int max_bonus;
> +#define MAX_BONUS              (max_bonus)
> +int interactive_delta;
> +#define INTERACTIVE_DELTA      (interactive_delta)
> +int max_sleep_avg;
> +#define MAX_SLEEP_AVG          (max_sleep_avg)
> +int starvation_limit;
> +#define STARVATION_LIMIT       (starvation_limit)
```

__read_mostly? Except for those variables which are being tweaked quite often by live scheduler code... (probably none)

This only concerns the case of CONFIG_DEBUGFS_SCHED being activated, but it may still be nice to have, I think, since scheduler stuff should rather be fast than slow.

But since it is a debugging-only option omitting __read_mostly may make sense...

Andreas Mohr

--
 VGER BF report: H 0.00129204

-
 To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org
 More majordomo info at <http://vger.kernel.org/majordomo-info.html>
 Please read the FAQ at <http://www.tux.org/lkml/>

Sep 05, 06 23:43

code-review.txt

Page 1/22

A Linux Kernel Mailing list style code review between Jake Moilanen and I.

From moilanen@austin.vnet Mon Jul 31 18:22:07 2006
 Subject: Re: [patch 0/8] [ANNOUNCE] genetic library w/ plugins for the CPU scheduler and anticipatory I/O scheduler
 From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
 To: philips@vnet.ibm.com

On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
 > These 8 patches implement an in Kernel genetic library with plugins for
 > both the anticipatory I/O scheduler and CPU scheduler.
 >
 > The goal of these patches is to improve performance of a system under
 > various workloads by changing static tunables that are tuned by a
 > developer into dynamic tunables that are mutated by an iterative genetic
 > algorithm.
 >
 > These patches apply against 2.6.18-rc3.

You did the correct way of doing patches by showing incremental updates and breaking patches up that way. However, for the genetic-lib stuff, we have to treat this as an out-of-tree patchset. That means making it as easy for "Joe Linux" to be able to apply and build this. So, consolidating it down to 3 patches, and turning everything on in the defconfig for supported arch (ppc, pseries, pmac, i386):

1.) Base genetic-lib w/ fingerprinting code included
 2.) AS plugin
 3.) 0(1) plugin

The current "plan" is to get fingerprinting code cleaned up. I'm going to release the genetic-lib w/ that code and AS. While I'm cleaning that up, I want you to get some performance numbers and see where everything is on the 0(1). If you can show performance not decreasing too much, I want you to submit the 0(1) plugin yourself as an announce...State that they depend on the genetic-lib code, and put links to the patches it depends on.

If you get the GUI piece working well, do a separate email w/ an announce about that too....

Ok...let's start the code review. :)

From moilanen@austin.vnet Tue Aug 1 15:05:53 2006
 Subject: Re: [patch 2/8] genetic-lib: add debugfs entries
 From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
 To: philips@vnet.ibm.com

On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
 > plain text document attachment (genetic-lib)
 > Features include:

While most mail agents handle this correct, make sure you do an "insert of text file". Don't do attachment.

> * Toggle gene mutations from being applied to a particular plugin via
 > /debug/genetic/<plugin>/defaults
 > * View the values of genes in each phenotype via
 > /debug/genetic/<plugin>/phenotypes/<phenotype>
 > * View stat averages via /debug/genetic/<plugin>/phenotype_average
 > * View general plugin statistics via /debug/genetic/<plugin>/stats
 >
 > A very simple TCL/TK GUI that parses and logs the phenotypes can be found here:
 > <http://ifup.org/git/?p=genetic-tools.git;a=blob;hb=HEAD;f=genetk>

While this should be a separate announce...make it so it's a direct download...remember, a lot of the Linux users are not savvy, so they don't know how to use GIT or even how to click the correct link to get a downloadable version.

> Signed-off-by: Brandon Philips <philips@vnet.ibm.com>

Since you're only here for a few months, I would use your home email for this.

```
> Index: linux-gl/lib/genetic-debug.c
> =====
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-gl/lib/genetic-debug.c      2006-07-14 16:45:44.000000000 -0500
> @@ -0,0 +1,153 @@
> +/*
> + * Genetic Algorithm Library Debugging Routines
> + *
```

I'm not crazy about this being it's own file. Put this at the bottom of genetic.c.

I like the idea of having this information exported for genetk and always exported. And since this has come around, it means that this information should not be in debugfs. Debugfs is meant for non-persistent information. (ie it's turned on `_only_` when someone wants to see debug information). This should really live in sysfs.

```
> + * (C) Copyright 2006 IBM
> + * (C) Copyright 2006 Brandon Philips <brandon@ifup.org>
> + *
> + * This program is free software; you can redistribute it and/or modify it
```

Tuesday September 05, 2006

1/68

```

> + * under the terms of version 2 of the GNU General Public License as published
> + * by the Free Software Foundation.
> + */
> +
> +#include <linux/debugfs.h>
> +#include <linux/genetic.h>
> +#include <linux/seq_file.h>
> +
> +struct dentry * genetic_tree_root = NULL;
> +
> +/**
> + * genetic_stat_show - generates /debug/genetic/<name>/stats
> + */

Linux style has comments looking like:

/* This has just recently moved to starting text on this line.
 * And not on this line
 */

(and no ** on the first line).

> +static int genetic_stat_show(struct seq_file *s, void *unused)
> +{
> +    genetic_t * genetic = (genetic_t *)s->private;
> +
> +    seq_printf(s, "name: %s\n", genetic->name);
> +    seq_printf(s, "generation_number: %ld\n", genetic->generation_number);
> +    seq_printf(s, "num_children: %ld\n", genetic->num_children);
> +    seq_printf(s, "child_life_time: %ld\n", genetic->child_life_time);
> +    seq_printf(s, "child_number: %ld\n", genetic->child_number);
> +
> +    return 0;
> +}
> +
> +static int genetic_stat_open(struct inode *inode, struct file *file)
> +{
> +    return single_open(file, genetic_stat_show, inode->u.generic_ip);
> +}
> +
> +static struct file_operations genetic_stat_operations = {
> +    .open      = genetic_stat_open,
> +    .read      = seq_read,
> +    .llseek    = seq_lseek,
> +    .release   = single_release,
> +};
> +
> +/**
> + * genetic_phenotype_average_show - /debug/genetic/<name>/phenotype_average
> + */
> +static int genetic_phenotype_average_show(struct seq_file *s, void *unused)
> +{
> +    genetic_t * genetic = (genetic_t *)s->private;
> +    struct list_head * p;
> +    phenotype_t * pt;
> +
> +    list_for_each(p, &genetic->phenotype) {
> +        pt = list_entry(p, phenotype_t, phenotype);
> +        seq_printf(s, "%s: %lld\n", pt->name, pt->avg_fitness);
> +    }
> +
> +    return 0;
> +}
> +
> +static int genetic_phenotype_average_open(struct inode *inode, struct file *file)
> +{
> +    return single_open(file, genetic_phenotype_average_show, inode->u.generic_ip);
> +}
> +
> +static struct file_operations genetic_phenotype_average_operations = {
> +    .open      = genetic_phenotype_average_open,
> +    .read      = seq_read,
> +    .llseek    = seq_lseek,
> +    .release   = single_release,
> +};
> +
> +
nit: typically single new line.

> +/**
> + * genetic_genes_show - /debug/genetic/<name>/gene
> + */
> +int genetic_generic_gene_show(struct seq_file *s, void *unused)
> +{
> +    int i;
> +    phenotype_t * pt = (phenotype_t *)s->private;
> +
> +    genetic_child_t * child = list_entry(pt->run_queue->next,
> +                                       genetic_child_t, list);
> +
> +    unsigned long * genes = (unsigned long *)child->genes;
> +
> +    for (i = 0; i < pt->num_genes; i++)
> +        seq_printf(s, "%s: %lu\n", child->gene_param[i].name, genes[i]);
> +
> +    return 0;
> +}
> +
> +static int genetic_generic_gene_open(struct inode *inode, struct file *file)

```

```

> +{
> +     phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
> +
> +     return single_open(file, pt->ops->gene_show, inode->u.generic_ip);
> +}
> +
> +static struct file_operations genetic_gene_operations = {
> +     .open           = genetic_generic_gene_open,
> +     .read           = seq_read,
> +     .llseek        = seq_lseek,
> +     .release        = single_release,
> +};
> +
> +
> +static struct dentry *genetic_create_tree(const char *name, struct dentry *parent)
> +{
> +     struct dentry *dir = NULL;
> +
> +     if (!genetic_tree_root) {
> +         genetic_tree_root = debugfs_create_dir("genetic", NULL);
> +         if (!genetic_tree_root)
> +             goto err;
> +     }
> +
> +     if (!parent) parent = genetic_tree_root;
> +
> +     dir = debugfs_create_dir(name, parent);
> +
> +err:
> +     return dir;
> +}
> +
> +#if GENETIC_DEBUG
> +/* Stores attributes into an array in the following format
> + * child_num fitness gene[0] gene[1] ... gene[num_genes-1]
> + * Add +1 to GENETIC_NUM_DEBUG_POINTS if add another dump_children
> + * call
> + */
> +void dump_children(phenotype_t * pt)
> +{
> +     int i, j;
> +     long * genes;
> +     unsigned long debug_size = pt->debug_size;
> +
> +     for (i = 0; i < pt->num_children; i++) {
> +         pt->debug_history[pt->debug_index++ % debug_size] = pt->child_ranking[i]->id;
> +         pt->debug_history[pt->debug_index++ % debug_size] = pt->child_ranking[i]->fitness;
> +
> +         genes = (long *)pt->child_ranking[i]->genes;
> +
> +         for (j = 0; j < pt->child_ranking[i]->num_genes; j++) {
> +             pt->debug_history[pt->debug_index++ % debug_size] = genes[j];
> +         }
> +     }
> +}
> +#else
> +void dump_children(phenotype_t * pt) {}
> +#endif /* GENETIC_DEBUG */
> +Index: linux-gf/lib/genetic.c
> +=====
> +--- linux-gf.orig/lib/genetic.c      2006-07-14 16:45:22.000000000 -0500
> ++++ linux-gf/lib/genetic.c          2006-07-14 16:45:44.000000000 -0500
> +@@ -38,6 +38,8 @@
> + #include <asm/string.h>
> + #include <asm/bug.h>
> +
> + #include "genetic-debug.c"
> +

```

Kernel guys hate this. Don't #include a c file. Build it.

```

> char genetic_lib_version[] = "0.3.1";
>
> int mutation_rate_change = GENETIC_DEFAULT_MUTATION_RATE_CHANGE;
> @@ -98,6 +100,20 @@
>

```

Make your patches w/ -p option. I can't tell what function this cooresponds to...So it's much more difficult figuring out what you are actually changing.

Make a .quiltrc. Here's mine:

```

moilanen@style:~$ cat .quiltrc
QUILT_DIFF_OPTS="-p"
QUILT_REFRESH_ARGS="--no-timestamps --diffstat"

```

```

>     INIT_LIST_HEAD(&genetic->phenotype);
>
> +     /* Setup debugfs */
> +     genetic->dir = genetic_create_tree(name, NULL);
> +     genetic->phenotypes_dir = genetic_create_tree("phenotypes", genetic->dir);
> +
> +     /* TODO add stack to the genetic track dentries for deallocation */
> +     debugfs_create_file("stats", S_IFREG|S_IRUGO, genetic->dir,
> +         genetic, &genetic_stat_operations);
> +
> +     debugfs_create_file("phenotype_average", S_IFREG|S_IRUGO, genetic->dir,

```

Sep 05, 06 23:43

code-review.txt

Page 4/22

```

> +         genetic, &genetic_phenotype_average_operations);
> +
> +     debugfs_create_bool("defaults", S_IWUSR|S_IFREG|S_IRUGO, genetic->dir,
> +         &genetic->defaults);
Does this reset the genes to their defaults??
> +
> +     return 0;
> + }
> +
> @@ -170,6 +186,11 @@
> +
> +     list_add_tail(&pt->phenotype, &genetic->phenotype);
> +
> +     if (ops->gene_show) {
> +         debugfs_create_file(name, S_IFREG|S_IRUGO,
> +             genetic->phenotypes_dir, pt, &genetic_gene_operations);
> +     }
> +     return 0;
> + }
> +
> @@ -554,19 +575,27 @@
> +         if (pt->ops->calc_post_fitness)
> +             pt->ops->calc_post_fitness(pt);
> +
> +         dump_children(pt);
> +
Wasn't this here before??
> +
> +     /* figure out top performers */
> +     genetic_split_performers(pt);
> +
> +     /* calc stats */
> +     genetic_calc_stats(pt);
> +
> +     dump_children(pt);
> +
And this one?
> +
> +     /* make some new children */
> +     if (pt->num_genes)
> +         pt->natural_selection(pt);
> +
> +     dump_children(pt);
> +
ditto.
> +
> +     /* mutate a couple of the next generation */
> +     genetic_mutate(pt);
> +
> +     dump_children(pt);
> +
ditto.
> +
> +     /* Move the new children still sitting in the finished queue to
> +         the run queue */
> +     tmp = pt->run_queue;
> +
> --
> +
From moilanen@austin.vnet Tue Aug 1 15:26:08 2006
Subject: Re: [patch 2/8] genetic-lib: add debugfs entries
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: Brandon Philips <philips@vnet.ibm.com>

On Tue, 2006-08-01 at 15:14 -0500, Brandon Philips wrote:
> On 13:21 Tue 01 Aug 2006, Jake Moilanen wrote:
> > On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
> > > plain text document attachment (genetic-lib)
> > > Features include:
> >
> > > While most mail agents handle this correct, make sure you do an "insert
> > > of text file". Don't do attachment.
> >
> > Weird, I used quilt mail to do this.
> >
> > > A very simple TCL/TK GUI that parses and logs the phenotypes can be found here:
> > > http://ifup.org/git/?p=genetic-tools.git;a=blob;hb=HEAD;f=genetk
> >
> > > While this should be a separate announce...make it so it's a direct
> > > download...remember, a lot of the Linux users are not savvy, so they
> > > don't know how to use GIT or even how to click the correct link to get a
> > > downloadable version.
> >
> > Good call.
> >
> > Index: linux-gl/lib/genetic.c
> > > =====
> > > --- linux-gl.orig/lib/genetic.c      2006-07-14 16:45:22.000000000 -0500
> > > +++ linux-gl/lib/genetic.c          2006-07-14 16:45:44.000000000 -0500
> > > @@ -38,6 +38,8 @@
> > > #include <asm/string.h>
> > > #include <asm/bug.h>

```

Sep 05, 06 23:43

code-review.txt

Page 5/22

```

> > >
> > > #include "genetic-debug.c"
> > > +
> > > +
> > >
> > Kernel guys hate this. Don't #include a c file. Build it.
> >
> > I was following the lead of Linus on this one.
> >
> > ./drivers/usb/host/uhci-debug.c
Linus did that...ugh...
> >
> > I will move it to the end of the file as you suggest though.
> >
> > > Make your patches w/ -p option. I can't tell what function this
> > > cooresponds to...So it's much more difficult figuring out what you are
> > > actually changing.
> > >
> > > Make a .quiltrc. Here's mine:
> > >
> > > moilanen@style:~$ cat .quiltrc
> > > QUILT_DIFF_OPTS="-p"
> > > QUILT_REFRESH_ARGS="--no-timestamps --diffstat"
> > >
> > Ah, good call.
> >
> > > INIT_LIST_HEAD(&genetic->phenotype);
> > >
> > > + /* Setup debugfs */
> > > + genetic->dir = genetic_create_tree(name, NULL);
> > > + genetic->phenotypes_dir = genetic_create_tree("phenotypes", genetic->dir);
> > > +
> > > + /* TODO add stack to the genetic track dentries for deallocation */
> > > + debugfs_create_file("stats", S_IFREG|S_IRUGO, genetic->dir,
> > > + genetic, &genetic_stat_operations);
> > > +
> > > + debugfs_create_file("phenotype_average", S_IFREG|S_IRUGO, genetic->dir,
> > > + genetic, &genetic_phenotype_average_operations);
> > > +
> > > + debugfs_create_bool("defaults", S_IWUSR|S_IFREG|S_IRUGO, genetic->dir,
> > > + &genetic->defaults);
> > >
> > > Does this reset the genes to their defaults??
> > >
> > Yes.
Does it turn the genetic library off?
> > > @@ -554,19 +575,27 @@
> > >         if (pt->ops->calc_post_fitness)
> > >             pt->ops->calc_post_fitness(pt);
> > >
> > > +         dump_children(pt);
> > > +
> > >
> > > Wasn't this here before??
> > >
> > > Yes, but I wanted to have the patches build at every step. And the
> > > second patch adds these functions. I guess I will fold those two
> > > patches together for now.
Ahh...Ok.
From moilanen@austin.vnet Tue Aug 1 16:39:29 2006
Subject: Re: [patch 2/8] genetic-lib: add debugfs entries
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: Brandon Philips <philips@vnet.ibm.com>
On Tue, 2006-08-01 at 16:24 -0500, Brandon Philips wrote:
> > On 13:42 Tue 01 Aug 2006, Jake Moilanen wrote:
> > > > Does this reset the genes to their defaults??
> > > >
> > > > Yes.
> > > >
> > > > It only bypasses the setting of the genes in genetic_switch_child.
So, this really is an enablement/disablement of the genetic-library for
this plugin. Probably should have a better name than "defaults". Like
"run", or "enabled"...
From moilanen@austin.vnet Tue Aug 1 15:07:55 2006
Subject: Re: [patch 3/8] genetic-lib: fingerprinting base
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: philips@vnet.ibm.com
On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
> > plain text document attachment (genetic-lib)
> > Creates a general framework for fingerprinting workloads and reintroducing
> > genes that were known to work well on that workload. For a detailed
> > introduction see Jake Moilanen's paper from OLS 2006.
> >
> > http://www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf pg. 173
> >
> > Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>
> > Signed-off-by: Brandon Philips <philips@vnet.ibm.com>
This was all prior code correct?
> > Index: linux-gl/lib/fingerprinting.c

```

Tuesday September 05, 2006

5/68

```

> -----
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-gl/lib/fingerprinting.c      2006-07-14 16:46:19.000000000 -0500
> @@ -0,0 +1,291 @@
> +static int create_fingerprint(struct fingerprint * fp)
> +{
> +    int numerical_fp = 0;
> +
> +    numerical_fp |= fp->type;
> +    numerical_fp <<= 1;
> +
> +    numerical_fp |= fp->pattern;
> +    numerical_fp <<= 1;
> +
> +    numerical_fp |= fp->size;
> +
> +    return numerical_fp;
> +}
> +
> +static long long get_top_fitness(phenotype_t * pt, struct fingerprint * fp)
> +{
> +    return pt->top_fitness[fp->type][fp->pattern][fp->size];
> +}
> +
> +
> +static int top_fitness_open(struct inode *inode, struct file *file)
> +{
> +    phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
> +
> +    return single_open(file, pt->ops->top_fitness_show, inode->u.generic_ip);
> +}
> +
> +static struct file_operations top_fitness_ops = {
> +    .open          = top_fitness_open,
> +    .read          = seq_read,
> +    .llseek       = seq_lseek,
> +    .release      = single_release,
> +};
> +
> +static int snapshot_open(struct inode *inode, struct file *file)
> +{
> +    phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
> +
> +    return single_open(file, pt->ops->snapshot_show, inode->u.generic_ip);
> +}
> +
> +static struct file_operations snapshot_ops = {
> +    .open          = snapshot_open,
> +    .read          = seq_read,
> +    .llseek       = seq_lseek,
> +    .release      = single_release,
> +};
> +
> +static int state_open(struct inode *inode, struct file *file)
> +{
> +    phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
> +
> +    return single_open(file, pt->ops->state_show, inode->u.generic_ip);
> +}
> +
> +static struct file_operations state_ops = {
> +    .open          = state_open,
> +    .read          = seq_read,
> +    .llseek       = seq_lseek,
> +    .release      = single_release,
> +};
> +
> +
> +int genetic_init_fingerprinting(phenotype_t * pt)
> +{
> +    int i, j, k;
> +    struct genetic_ops * ops = pt->ops;
> +    int num_genes = pt->num_genes;
> +
> +    if (num_genes) {
> +
> +        pt->fp = (struct fingerprint *)kmalloc(
> +            sizeof(struct fingerprint), GFP_KERNEL);
> +
> +        if (!pt->fp) {
> +            printk(KERN_ERR "genetic_register_phenotype: not enough"
> +                "memory\n");
> +            return -ENOMEM;
> +        }
> +
> +        reset_fp(pt->fp);
> +
> +        pt->fp_ss = (struct fp_snapshot *)kmalloc(
> +            sizeof(struct fp_snapshot), GFP_KERNEL);
> +
> +        if (!pt->fp_ss) {
> +            printk(KERN_ERR "genetic_register_phenotype: not enough"
> +                "memory\n");
> +            return -ENOMEM;
> +        }
> +
> +        reset_fp_snapshot(pt->fp_ss);
> +
> +        pt->top_child = (unsigned long ***)kmalloc(
> +            sizeof(unsigned long ***) * 2, GFP_KERNEL);

```

```

> +
> +     if (!pt->top_child) {
> +         printk(KERN_ERR "genetic_register_phenotype: not enough"
> +             "memory\n");
> +         return -ENOMEM;
> +     }
> +
> +     for (i = 0; i < 2; i++) {
> +         pt->top_child[i] = (unsigned long **)kmallocc(
> +             sizeof(unsigned long **) * 2,
> +             GFP_KERNEL);
> +
> +         if (!pt->top_child[i]) {
> +             printk(KERN_ERR "genetic_register_phenotype:\
> +                 not enough memory\n");
> +             return -ENOMEM;
> +         }
> +
> +         for (j = 0; j < 2; j++) {
> +             pt->top_child[i][j] = (unsigned long *)kmallocc(
> +                 sizeof(unsigned long *) * 2,
> +                 GFP_KERNEL);
> +
> +             if (!pt->top_child[i][j]) {
> +                 printk(KERN_ERR "genetic_register_phenotype: not enough memory\n");
> +                 return -ENOMEM;
> +             }
> +
> +             for (k = 0; k < 2; k++) {
> +                 pt->top_child[i][j][k] = (unsigned long)ops->create_top_genes(pt);
> +                 if (!pt->top_child[i][j][k])
> +                     return -ENOMEM;
> +             }
> +         }
> +     }
> + } /* if (num_genes) */
> +
> + pt->top_fitness = (long long ***)kmallocc(sizeof(long long ***) * 2, GFP_KERNEL);
> + if (!pt->top_fitness) {
> +     printk(KERN_ERR "genetic_register_phenotype: not enough"
> +         "memory\n");
> +     return -ENOMEM;
> + }
> +
> + for (i = 0; i < 2; i++) {
> +     pt->top_fitness[i] = (long long **)kmallocc(sizeof(long long **) * 2, GFP_KERNEL);
> +     if (!pt->top_fitness[i]) {
> +         printk(KERN_ERR "genetic_register_phenotype: not"
> +             "enough memory\n");
> +         return -ENOMEM;
> +     }
> +
> +     for (j = 0; j < 2; j++) {
> +         pt->top_fitness[i][j] = (long long *)kmallocc(
> +             sizeof(long long *) * 2,
> +             GFP_KERNEL);
> +
> +         if (!pt->top_fitness[i][j]) {
> +             printk(KERN_ERR "genetic_register_phenotype: "
> +                 "not enough memory\n");
> +             return -ENOMEM;
> +         }
> +
> +         for (k = 0; k < 2; k++) {
> +             pt->top_fitness[i][j][k] = 0;
> +         }
> +     }
> + }
> +
> + pt->last_fingerprint = 0;
> +
> + if (pt->genetic->fingerprinting_dir) {
> +     pt->fp_dir = genetic_create_tree(pt->name,
> +         pt->genetic->fingerprinting_dir);
> +
> +     if (ops->top_fitness_show)
> +         debugfs_create_file("top_fitness", S_IFREG|S_IRUGO,
> +             pt->fp_dir, pt, &top_fitness_ops);
> +
> +     if (ops->snapshot_show)
> +         debugfs_create_file("snapshot", S_IFREG|S_IRUGO,
> +             pt->fp_dir, pt, &snapshot_ops);
> +
> +     if (ops->state_show)
> +         debugfs_create_file("state", S_IFREG|S_IRUGO,
> +             pt->fp_dir, pt, &state_ops);
> + }
> +
> + return 0;
> + }
> +
> + static void decay_fitness(phenotype_t * pt, struct fingerprint * fp)
> + {
> +     long long fitness;
> +     long dummy;
> +
> +     fitness = get_top_fitness(pt, fp);
> +
> +     /* reduce the fitness to eventually get new genes in */
> +     fitness *= FP_DECAY;

```

```

> +   divll(&fitness, 100, &dummy);
> +
> +   pt->top_fitness[fp->type][fp->pattern][fp->size] = fitness;
> +}
> +
> +static void update_phenotype_top_performer(phenotype_t * pt, struct fingerprint * fp)
> +{
> +   long long top_fitness;
> +   unsigned long * genes;
> +   long long * avg_genes;
> +   long dummy;
> +   int i, j;
> +
> +   /* Decay the top fitness so not to have a fluke and have a
> +    * high set which are less than optimal. So decay the top
> +    * fitness so eventually these genes are phased out.
> +    */
> +   decay_fitness(pt, fp);
> +
> +   top_fitness = get_top_fitness(pt, fp);
> +
> +   if (pt->last_gen_avg_fitness >= top_fitness) {
> +
> +       pt->top_fitness[fp->type][fp->pattern][fp->size] = pt->last_gen_avg_fitness;
> +
> +       /* We don't need to track this if there's no genes! */
> +       if (!pt->num_genes)
> +           return;
> +
> +       avg_genes = (long long *)kmalloc(sizeof(long long) * pt->num_genes, GFP_KERNEL);
> +       if (!avg_genes) {
> +           printk(KERN_ERR "update_top_performers: unable to alloc space\n");
> +           return;
> +       }
> +
> +       memset(avg_genes, 0, sizeof(long long) * pt->num_genes);
> +
> +       for (i = 0; i < pt->num_genes; i++) {
> +           for (j = 0; j < pt->num_children; j++) {
> +               genes = pt->child_ranking[j]->genes;
> +               avg_genes[i] += genes[i];
> +           }
> +       }
> +
> +       for (j = 0; j < pt->num_genes; j++)
> +           divll(&avg_genes[j], pt->num_children, &dummy);
> +
> +       genes = (unsigned long *)pt->top_child[fp->type][fp->pattern][fp->size];
> +       for (j = 0; j < pt->num_genes; j++)
> +           genes[j] = avg_genes[j];
> +
> +       kfree(avg_genes);
> +   }
> +}
> +
> +static void update_top_performers(phenotype_t * master)
> +{
> +   phenotype_t * pt;
> +   struct list_head * p;
> +
> +   list_for_each(p, &master->genetic->phenotype) {
> +       pt = list_entry(p, phenotype_t, phenotype);
> +
> +       if (master->uid & pt->uid && master->uid != pt->uid) {
> +           update_phenotype_top_performer(pt, master->fp);
> +       }
> +   }
> +   update_phenotype_top_performer(master, master->fp);
> +}
> +
> +static void reintroduce_genes(phenotype_t * master)
> +{
> +   struct fingerprint * fp = master->fp;
> +   phenotype_t * pt;
> +   unsigned long * top_genes;
> +   unsigned long * genes;
> +   struct list_head * p;
> +   int i;
> +
> +   list_for_each(p, &master->genetic->phenotype) {
> +       pt = list_entry(p, phenotype_t, phenotype);
> +
> +       if (pt->num_genes) {
> +
> +           /* Do this more intelligently, so can have n-points on
> +            * the fingerprint */
> +           /* just take the first one */
> +           top_genes = (unsigned long *)pt->top_child[fp->type][fp->pattern][fp->size];
> +           genes = pt->child_ranking[0]->genes;
> +           for (i = 0; i < pt->num_children; i++)
> +               genes[i] = top_genes[i];
> +       }
> +   }
> +}
> +
> +Index: linux-gl/include/linux/fingerprinting.h
> +-----
> +--- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-gl/include/linux/fingerprinting.h 2006-07-14 16:46:19.000000000 -0500
> @@ -0,0 +1,125 @@

```

Sep 05, 06 23:43

code-review.txt

Page 9/22

```

> #ifndef __LINUX_FINGERPRINTING_H
> #define __LINUX_FINGERPRINTING_H
> +
> +/*
> + * include/linux/fingerprinting.h
> + *
> + * Jake Moilanen <moilanen@austin.ibm.com>
> + * Copyright (C) 2006 IBM
> + *
> + * I/O Workload Fingerprinting
> + *
> + * This program is free software; you can redistribute it and/or modify it
> + * under the terms of version 2 of the GNU General Public License as published
> + * by the Free Software Foundation.
> +*/
> +
> +#include <linux/types.h>
> +#include <linux/bio.h>
> +
> +#define FP_TYPE_READ 0
> +#define FP_TYPE_WRITE 1
> +#define FP_PATTERN_SEQ 0
> +#define FP_PATTERN_RAND 1
> +#define FP_SIZE_SMALL 0
> +#define FP_SIZE_LARGE 1
> +#define FP_NUM_POINTS (2 * 2 * 2)
> +
> +struct fingerprint {
> +    __u8 type;
> +    __u8 pattern;
> +    __u8 size;
> +};
> +
> +struct fp_snapshot {
> +    /* type */
> +    unsigned long reads;
> +    unsigned long writes;
> +    /* pattern */
> +    unsigned long head_pos;
> +    unsigned long avg_dist;
> +    /* size */
> +    unsigned long avg_size;
> +};
> +
> +/* Number of reads/writes before classified as read */
> +#define FP_CLASS_READ_WRITE_RATIO 2
> +
> +/* Number of sectors before pattern is random */
> +#define FP_CLASS_PATTERN_RAND 25
> +
> +/* Number of sectors before size is large */
> +#define FP_CLASS_SIZE_LARGE 8
> +
> +extern void update_fp_snapshot(struct bio * bio);
> +extern void calc_fp(struct fingerprint * fp, struct fp_snapshot * fp_ss);
> +extern void reset_fp_snapshot(struct fp_snapshot * ss);
> +extern void reset_fp(struct fingerprint * fp);
> +extern void consolidate_fp_snapshot(struct fp_snapshot * master, struct fp_snapshot * instance);
> +extern int fingerprint_state_show(struct seq_file *s, void *unused);
> +extern int fingerprint_snapshot_show(struct seq_file *s, void *unused);
> +extern int fingerprint_top_fitness_show(struct seq_file *s, void *unused);
> +
> +/* XXX do this more intelligently */
> +#ifndef DIVLL_OP
> +#define DIVLL_OP
> +#if BITS_PER_LONG >= 64
> +
> +static inline void divll(long long *n, long div, long *rem)
> +{
> +    *rem = *n % div;
> +    *n /= div;
> +}
> +
> +#else
> +
> +static inline void divl(int32_t high, int32_t low,
> +    int32_t div,
> +    int32_t *q, int32_t *r)
> +{
> +    int64_t n = (u_int64_t)high << 32 | low;
> +    int64_t d = (u_int64_t)div << 31;
> +    int32_t ql = 0;
> +    int c = 32;
> +    while (n > 0xffffffff) {
> +        ql <= 1;
> +        if (n >= d) {
> +            n -= d;
> +            ql |= 1;
> +        }
> +        d >>= 1;
> +        c--;
> +    }
> +    ql <= c;
> +    if (n) {
> +        low = n;
> +        *q = ql | (low / div);
> +        *r = low % div;
> +    } else {
> +        *r = 0;
> +        *q = ql;

```

```

> +     }
> +     return;
> +}
> +
> +static inline void divll(long long *n, long div, long *rem)
> +{
> +     int32_t low, high;
> +     low = *n & 0xffffffff;
> +     high = *n >> 32;
> +     if (high) {
> +         int32_t high1 = high % div;
> +         int32_t low1 = low;
> +         high /= div;
> +         divl(high1, low1, div, &low, (int32_t *)rem);
> +         *n = (int64_t)high << 32 | low;
> +     } else {
> +         *n = low / div;
> +         *rem = low % div;
> +     }
> +}
> +#endif
> +
> +#endif /* #ifndef divll */
> +
> +#endif /* __LINUX_FINGERPRINTINT_H */
> index: linux-g1/lib/genetic.c
> =====
> --- linux-g1.orig/lib/genetic.c      2006-07-14 16:45:44.000000000 -0500
> +++ linux-g1/lib/genetic.c          2006-07-14 16:46:19.000000000 -0500
> @@ -40,6 +40,11 @@
>
> #include "genetic-debug.c"
>
> #ifdef CONFIG_FINGERPRINTING
> #include <linux/fingerprinting.h>
> #include "fingerprinting.c"
> #endif
>
> char genetic_lib_version[] = "0.3.1";
>
> int mutation_rate_change = GENETIC_DEFAULT_MUTATION_RATE_CHANGE;
> @@ -92,6 +97,9 @@
>     genetic->generation_number = 1;
>     genetic->child_number = 0;
>     genetic->defaults = 0;
> #ifdef CONFIG_FINGERPRINTING
>     genetic->fingerprinting = fingerprinting;
> #endif
>
>     /* Setup how long each child has to live */
>     init_timer(&genetic->timer);
> @@ -104,6 +112,11 @@
>     genetic->dir = genetic_create_tree(name, NULL);
>     genetic->phenotypes_dir = genetic_create_tree("phenotypes", genetic->dir);
>
> #ifdef CONFIG_FINGERPRINTING
> +     if (fingerprinting)
> +         genetic->fingerprinting_dir = genetic_create_tree("fingerprinting", genetic->dir);
> #endif
>
>     /* TODO add stack to the genetic track dentries for deallocation */
>     debugfs_create_file("stats", S_IFREG|S_IRUGO, genetic->dir,
>         genetic, &genetic_stat_operations);
> @@ -179,6 +192,13 @@
>
>     pt->top_fitness = 0;
>
> #ifdef CONFIG_FINGERPRINTING
> +     if (genetic->fingerprinting) {
> +         if ((rc = genetic_init_fingerprinting(pt)) < 0)
> +             return rc;
> +     }
> #endif
>
>     /* Create some children */
>     rc = genetic_create_children(pt);
>     if (rc)
> @@ -478,6 +498,10 @@
>     long long tmp_fitness;
>     long dummy;
>     int i = 0;
> #ifdef CONFIG_FINGERPRINTING
>     int fp = in_pt->genetic->fingerprinting;
>     int numerical_fp;
> #endif
>
>     /* On a general phenotype, need to look at other metrics since
>      * the fitness is normalized. It always average the same. It
> @@ -485,7 +509,15 @@
>      */
>     if (in_pt->ops->calc_post_fitness) {
> #ifdef CONFIG_FINGERPRINTING
> +         if (fp)
> +             numerical_fp = create_fingerprint(in_pt->fp);
> +
> +         /* do we want this???? */
> +         if ((fp && (in_pt->last_fingerprint == numerical_fp)) || !fp) {
> #else
> +         if (1) {

```

Sep 05, 06 23:43

code-review.txt

Page 11/22

```

> +#endif
>
> #ifdef GENETIC_DEBUG_VERBOSE
>     printk(KERN_INFO "genetic_calc_stats() for %s\n", in_pt->name);
> @@ -591,6 +623,29 @@
>
>         dump_children(pt);
>
> +#ifdef CONFIG_FINGERPRINTING
> +     if (pt->ops->get_fingerprint) {
> +
> +         pt->ops->get_fingerprint(pt);
> +         reset_fp_snapshot(pt->fp_ss);
> +
> +         /* See if this generation was a top performer
> +          * for the current workload.
> +          * Do this after natural selection to get rid
> +          * of the bad apples
> +          */
> +         update_top_performers(pt);
> +
> +         /* We know the workload, lets put some known
> +          * good genes back in */
> +         reintroduce_genes(pt);
> +
> +         pt->last_fingerprint = create_fingerprint(pt->fp);
> +     }
> +
> +     dump_children(pt);
> +#endif
>
>     /* mutate a couple of the next generation */
>     genetic_mutate(pt);
>
>
> --
>

```

From moilanen@austin.vnet Tue Aug 1 15:37:21 2006
Subject: Re: [patch 3/8] genetic-lib: fingerprinting base
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: Brandon Philips <philips@vnet.ibm.com>

```

On Tue, 2006-08-01 at 15:15 -0500, Brandon Philips wrote:
> On 13:23 Tue 01 Aug 2006, Jake Moilanen wrote:
> > On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
> > > plain text document attachment (genetic-lib)
> > > Creates a general framework for fingerprinting workloads and reintroducing
> > > genes that were known to work well on that workload. For a detailed
> > > introduction see Jake Moilanen's paper from OLS 2006.
> > >
> > > http://www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf pg. 173
> > >
> > > Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>
> > > Signed-off-by: Brandon Philips <philips@vnet.ibm.com>
> >
> > This was all prior code correct?
>
> Yes, just moved about and put into its own file.
>
> I take it this is bad too:
>
> #ifdef CONFIG_FINGERPRINTING
> #include <linux/fingerprinting.h>
> #include "fingerprinting.c"
> #endif

```

Keep it in genetic.c for now...If it starts looking too messy, it might need it's own file.

From moilanen@austin.vnet Tue Aug 1 15:11:58 2006
Subject: Re: [patch 5/8] block: General purpose routines to fingerprint disk IO
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: philips@vnet.ibm.com

```

On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
> plain text document attachment (genetic-io)
> Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>

```

Normally, 4 and 5 should probably be combined.

```

> Index: linux-rc/block/genhd.c
> =====
> --- linux-rc.orig/block/genhd.c      2006-07-31 12:18:46.000000000 -0400
> +++ linux-rc/block/genhd.c          2006-07-31 12:18:54.000000000 -0400
> @@ -29,6 +29,8 @@
>     char name[16];
> } *major_names[BLKDEV_MAJOR_HASH_SIZE];
>
> +LIST_HEAD(gendisks);
> +
> /* index in the above - for now: assume no multimajor ranges */
> static inline int major_to_index(int major)
> {
> @@ -387,19 +389,22 @@
>     jiffies_to_msecs(disk_stat_read(disk, io_ticks)),
>     jiffies_to_msecs(disk_stat_read(disk, time_in_queue)));
> }
> +

```

```

> #ifdef CONFIG_FINGERPRINTING
> static ssize_t disk_fp_read(struct gendisk * disk, char *page)
> {
>     return sprintf(page, "reads: %llx\n"
>         "writes: %llx\n"
>         "head_pos: %llx\n"
>         "avg_dist: %llx\n",
>         "avg_size: %llx\n",
>     return sprintf(page, "reads: %lld\n"
>         "writes: %lld\n"
>         "head_pos: %lld\n"
>         "avg_dist: %lld\n",
>         "avg_size: %lld\n",
>         (unsigned long long)disk->fp_ss->reads,
>         (unsigned long long)disk->fp_ss->writes,
>         (unsigned long long)disk->fp_ss->head_pos,
>         (unsigned long long)disk->fp_ss->avg_dist,
>         (unsigned long long)disk->fp_ss->avg_size);
> }
> #endif
>
> static struct disk_attribute disk_attr_uevent = {
>     .attr = { .name = "uevent", .mode = S_IWUSR },
> @@ -425,10 +430,13 @@
>     .attr = { .name = "stat", .mode = S_IRUGO },
>     .show = disk_stats_read
> };
>
> #ifdef CONFIG_FINGERPRINTING
> static struct disk_attribute disk_attr_fp = {
>     .attr = { .name = "fp", .mode = S_IRUGO },
>     .show = disk_fp_read
> };
> #endif
>
> static struct attribute * default_attrs[] = {
>     &disk_attr_uevent.attr,
> @@ -437,7 +445,9 @@
>     &disk_attr_removable.attr,
>     &disk_attr_size.attr,
>     &disk_attr_stat.attr,
> #ifdef CONFIG_FINGERPRINTING
>     &disk_attr_fp.attr,
> #endif
>     NULL,
> };
>
> @@ -446,6 +456,7 @@
>     struct gendisk *disk = to_disk(kobj);
>     kfree(disk->random);
>     kfree(disk->part);
> +     list_del(&disk->gendisks);
>     free_disk_stats(disk);
>     kfree(disk);
> }
> @@ -646,6 +657,7 @@
>     kobj_set_kset_s(disk, block_subsys);
>     kobject_init(&disk->kobj);
>     rand_initialize_disk(disk);
> +     list_add_tail(&disk->gendisks, &gendisks);
> }
>
>     disk->fp_ss = kmalloc(sizeof(struct fp_snapshot), GFP_KERNEL);
> Index: linux-rc/block/ll_rw_blk.c
> =====
> --- linux-rc.orig/block/ll_rw_blk.c      2006-07-31 12:18:46.000000000 -0400
> +++ linux-rc/block/ll_rw_blk.c          2006-07-31 12:18:54.000000000 -0400
> @@ -21,6 +21,7 @@
> #include <linux/string.h>
> #include <linux/init.h>
> #include <linux/bootmem.h> /* for max_pfn/max_low_pfn */
> #include <linux/genetic.h>
> #include <linux/completion.h>
> #include <linux/slab.h>
> #include <linux/swap.h>
> @@ -2608,6 +2609,141 @@
>     __elv_add_request(q, req, ELEVATOR_INSERT_SORT, 0);
> }
>
> #ifdef CONFIG_GENETIC_IOSCHED_AS
> +extern struct list_head gendisks;
> +
> +void disk_stats_snapshot(phenotype_t * pt)
> +{
> +     struct list_head * d;
> +     struct gendisk *disk;
> +     struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)pt->child_ranking[0]->stats_snapshot;
> +
> +     memset(ss, 0, sizeof(struct disk_stats_snapshot));
> +
> +     list_for_each(d, &gendisks) {
> +         disk = list_entry(d, struct gendisk, gendisks);
> +
> +         disk_round_stats(disk);
> +
> +         ss->reads += disk_stat_read(disk, ios[READ]);
> +         ss->writes += disk_stat_read(disk, ios[WRITE]);
> +         ss->read_sectors += disk_stat_read(disk, sectors[READ]);
> +         ss->write_sectors += disk_stat_read(disk, sectors[WRITE]);
> +         ss->time_in_queue += disk_stat_read(disk, time_in_queue);

```

```

> +    }
> +}
> +
> +long long disk_num_ops_calc_fitness(genetic_child_t * child)
> +{
> +    struct list_head * d;
> +    struct gendisk *disk;
> +    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)child->stats_snapshot;
> +    long long reads = 0;
> +    long long writes = 0;
> +
> +    list_for_each(d, &gendisks) {
> +        disk = list_entry(d, struct gendisk, gendisks);
> +
> +        disk_round_stats(disk);
> +
> +        reads += disk_stat_read(disk, ios[READ]);
> +        writes += disk_stat_read(disk, ios[WRITE]);
> +    }
> +
> +    reads -= ss->reads;
> +    writes -= ss->writes;
> +
> +    return reads + writes;
> +}
> +
> +long long disk_throughput_calc_fitness(genetic_child_t * child)
> +{
> +    struct list_head * d;
> +    struct gendisk *disk;
> +    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)child->stats_snapshot;
> +    long long read_sectors = 0;
> +    long long write_sectors = 0;
> +
> +    list_for_each(d, &gendisks) {
> +        disk = list_entry(d, struct gendisk, gendisks);
> +
> +        disk_round_stats(disk);
> +
> +        read_sectors += disk_stat_read(disk, sectors[READ]);
> +        write_sectors += disk_stat_read(disk, sectors[WRITE]);
> +    }
> +
> +    read_sectors -= ss->read_sectors;
> +    write_sectors -= ss->write_sectors;
> +
> +    return read_sectors + write_sectors;
> +}
> +
> +long long disk_latency_calc_fitness(genetic_child_t * child)
> +{
> +    struct list_head * d;
> +    struct gendisk *disk;
> +    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)child->stats_snapshot;
> +    long long time_in_queue = 0;
> +
> +    list_for_each(d, &gendisks) {
> +        disk = list_entry(d, struct gendisk, gendisks);
> +
> +        disk_round_stats(disk);
> +
> +        time_in_queue += disk_stat_read(disk, time_in_queue);
> +    }
> +
> +    time_in_queue = -(time_in_queue - ss->time_in_queue);
> +
> +    return time_in_queue;
> +}
> +
> +#ifdef CONFIG_FINGERPRINTING
> +
> +void disk_update_fingerprint(phenotype_t * pt)
> +{
> +    struct list_head * d;
> +    struct gendisk *disk;
> +
> +    BUG_ON(!pt->fp_ss);
> +
> +    /* tally up all the other disk snapshots */
> +    list_for_each(d, &gendisks) {
> +        disk = list_entry(d, struct gendisk, gendisks);
> +
> +        consolidate_fp_snapshot(pt->fp_ss, disk->fp_ss);
> +
> +        /* reset it for the next generation */
> +        reset_fp_snapshot(disk->fp_ss);
> +    }
> +}
> +
> +void disk_get_fingerprint(phenotype_t * pt)
> +{
> +    struct list_head * d;
> +    struct gendisk *disk;
> +
> +    BUG_ON(!pt->fp_ss);
> +
> +    /* tally up all the other disk snapshots */
> +    list_for_each(d, &gendisks) {
> +        disk = list_entry(d, struct gendisk, gendisks);

```

Sep 05, 06 23:43

code-review.txt

Page 14/22

```

> +
> +     consolidate_fp_snapshot(pt->fp_ss, disk->fp_ss);
> +
> +     /* reset it for the next generation */
> +     reset_fp_snapshot(disk->fp_ss);
> + }
> +
> +     calc_fp(pt->fp, pt->fp_ss);
> +}
> +
> +#endif /* CONFIG_FINGERPRINTING */
> +
> +#endif /* GENETIC_IOSCHED_AS */
> +
> + /*
> +  * disk_round_stats()      - Round off the performance stats on a struct
> +  * disk_stats.
> + Index: linux-rc/include/linux/genhd.h
> + =====
> + --- linux-rc.orig/include/linux/genhd.h      2006-07-31 12:18:46.000000000 -0400
> + +++ linux-rc/include/linux/genhd.h          2006-07-31 12:18:54.000000000 -0400
> + @@ -124,6 +124,7 @@
> +     atomic_t sync_io;          /* RAID */
> +     unsigned long stamp;
> +     int in_flight;
> + struct list_head gendisks;
> + #ifdef CONFIG_SMP
> +     struct disk_stats *dkstats;
> + #else
> + Index: linux-rc/include/linux/blkdev.h
> + =====
> + --- linux-rc.orig/include/linux/blkdev.h      2006-07-31 12:14:46.000000000 -0400
> + +++ linux-rc/include/linux/blkdev.h          2006-07-31 12:18:54.000000000 -0400
> + @@ -833,12 +833,23 @@
> +     _res; \
> + } \
> + }
> + #endif
> + #endif
> +
> + #define MODULE_ALIAS_BLOCKDEV(major,minor) \
> +     MODULE_ALIAS("block-major-" __stringify(major) "-" __stringify(minor))
> + #define MODULE_ALIAS_BLOCKDEV_MAJOR(major) \
> +     MODULE_ALIAS("block-major-" __stringify(major) "-*")
> +
> + #ifdef CONFIG_GENETIC_IOSCHED_AS
> + +
> + +struct disk_stats_snapshot
> + +{
> + +     unsigned long reads;
> + +     unsigned long writes;
> + +     unsigned long read_sectors;
> + +     unsigned long write_sectors;
> + +     unsigned long time_in_queue;
> + +};
> + #endif /* CONFIG_GENETIC_IOSCHED_AS */
> +
> + #endif
> +
> + --
> +

```

From moilanen@austin.vnet Tue Aug 1 15:50:41 2006
Subject: Re: [patch 7/8] scheduler: Make tuning knobs global variables
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: philips@vnet.ibm.com

On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
> plain text document attachment (genetic-cpu)
> This patch replaces all #define "tuning knobs" in the scheduler with global
> variables that can be later manipulated by a genetic library plugin for the
> scheduler.

This looks good...Though, I haven't had a chance to go through the
variables, and make sure that all of them should be tunable.

```

> Signed-off-by: Brandon Philips <philips@vnet.ibm.com>
> Index: linux-rc/kernel/sched.c
> =====
> + --- linux-rc.orig/kernel/sched.c      2006-07-31 12:14:45.000000000 -0400
> + +++ linux-rc/kernel/sched.c          2006-07-31 12:19:06.000000000 -0400
> + @@ -87,18 +87,31 @@
> +  * default timeslice is 100 msecs, maximum timeslice is 800 msecs.
> +  * Timeslices get refilled after they expire.
> +  */
> + #define MIN_TIMESLICE          max(5 * HZ / 1000, 1)
> + #define DEF_TIMESLICE          (100 * HZ / 1000)
> + #define ON_RUNQUEUE_WEIGHT    30
> + #define CHILD_PENALTY         95
> + #define PARENT_PENALTY        100
> + #define EXIT_WEIGHT           3
> + #define PRIO_BONUS_RATIO       25
> + #define MAX_BONUS              (MAX_USER_PRIO * PRIO_BONUS_RATIO / 100)
> + #define INTERACTIVE_DELTA     2
> + #define MAX_SLEEP_AVG          (DEF_TIMESLICE * MAX_BONUS)
> + #define STARVATION_LIMIT      (MAX_SLEEP_AVG)
> + #define NS_MAX_SLEEP_AVG       (JIFFIES_TO_NS(MAX_SLEEP_AVG))
> + #define DEFAULT_MIN_TIMESLICE max(5 * HZ / 1000, 1)
> + #define DEFAULT_DEF_TIMESLICE (100 * HZ / 1000)
> + #define DEFAULT_ON_RUNQUEUE_WEIGHT 30

```

Sep 05, 06 23:43

code-review.txt

Page 15/22

```

> #define DEFAULT_CHILD_PENALTY          95
> #define DEFAULT_PARENT_PENALTY        100
> #define DEFAULT_EXIT_WEIGHT            3
> #define DEFAULT_PRIO_BONUS_RATIO      25
> #define DEFAULT_MAX_BONUS              (MAX_USER_PRIO * DEFAULT_PRIO_BONUS_RATIO / 100)
> #define DEFAULT_INTERACTIVE_DELTA     2
> #define DEFAULT_MAX_SLEEP_AVG         (DEFAULT_DEF_TIMESLICE * DEFAULT_MAX_BONUS)
> #define DEFAULT_STARVATION_LIMIT      (DEFAULT_MAX_SLEEP_AVG)
> #define DEFAULT_NS_MAX_SLEEP_AVG     (JIFFIES_TO_NS(DEFAULT_MAX_SLEEP_AVG))
> +
> +unsigned long min_timeslice;
> +unsigned long def_timeslice;
> +unsigned long on_runqueue_weight;
> +unsigned long child_penalty;
> +unsigned long parent_penalty;
> +unsigned long exit_weight;
> +unsigned long prio_bonus_ratio;
> +unsigned long max_bonus;
> +unsigned long interactive_delta;
> +unsigned long max_sleep_avg;
> +unsigned long starvation_limit;
> +unsigned long ns_max_sleep_avg;
> +
> /*
> * If a task is 'interactive' then we reinsert it in the active
> @@ -108,7 +121,7 @@
> *
> * This part scales the interactivity limit depending on niceness.
> *
> * We scale it linearly, offset by the INTERACTIVE_DELTA delta.
> * We scale it linearly, offset by the interactive_delta delta.
> * Here are a few examples of different nice levels:
> *
> * TASK_INTERACTIVE(-20): [1,1,1,1,1,1,1,1,1,0,0]
> @@ -129,33 +142,33 @@
> */
>
> #define CURRENT_BONUS(p) \
> - (NS_TO_JIFFIES((p)->sleep_avg) * MAX_BONUS / \
> - MAX_SLEEP_AVG)
> + (NS_TO_JIFFIES((p)->sleep_avg) * max_bonus / \
> + max_sleep_avg)
>
> #define GRANULARITY (10 * HZ / 1000 ? : 1)
>
> #ifdef CONFIG_SMP
> #define TIMESLICE_GRANULARITY(p) (GRANULARITY * \
> - (1 << (((MAX_BONUS - CURRENT_BONUS(p)) ? : 1) - 1)) * \
> + (1 << (((max_bonus - CURRENT_BONUS(p)) ? : 1) - 1)) * \
> + num_online_cpus())
> #else
> #define TIMESLICE_GRANULARITY(p) (GRANULARITY * \
> - (1 << (((MAX_BONUS - CURRENT_BONUS(p)) ? : 1) - 1)))
> + (1 << (((max_bonus - CURRENT_BONUS(p)) ? : 1) - 1)))
> #endif
>
> #define SCALE(v1,v1_max,v2_max) \
> (v1) * (v2_max) / (v1_max)
>
> #define DELTA(p) \
> - (SCALE(TASK_NICE(p) + 20, 40, MAX_BONUS) - 20 * MAX_BONUS / 40 + \
> - INTERACTIVE_DELTA)
> + (SCALE(TASK_NICE(p) + 20, 40, max_bonus) - 20 * max_bonus / 40 + \
> + interactive_delta)
>
> #define TASK_INTERACTIVE(p) \
> ((p)->prio <= (p)->static_prio - DELTA(p))
>
> #define INTERACTIVE_SLEEP(p) \
> - (JIFFIES_TO_NS(MAX_SLEEP_AVG * \
> - (MAX_BONUS / 2 + DELTA((p)) + 1) / MAX_BONUS - 1))
> + (JIFFIES_TO_NS(max_sleep_avg * \
> + (max_bonus / 2 + DELTA((p)) + 1) / max_bonus - 1))
>
> #define TASK_PREEMPTS_CURR(p, rq) \
> ((p)->prio < (rq)->curr->prio)
> @@ -170,14 +183,14 @@
> */
>
> #define SCALE_PRIO(x, prio) \
> - max(x * (MAX_PRIO - prio) / (MAX_USER_PRIO / 2), MIN_TIMESLICE)
> + max(x * (MAX_PRIO - prio) / (MAX_USER_PRIO / 2), min_timeslice)
>
> static unsigned int static_prio_timeslice(int static_prio)
> {
> - if (static_prio < NICE_TO_PRIO(0))
> - return SCALE_PRIO(DEF_TIMESLICE * 4, static_prio);
> + return SCALE_PRIO(def_timeslice * 4, static_prio);
> - else
> - return SCALE_PRIO(DEF_TIMESLICE, static_prio);
> + return SCALE_PRIO(def_timeslice, static_prio);
> }
>
> static inline unsigned int task_timeslice(struct task_struct *p)
> @@ -699,7 +712,7 @@
> * __normal_prio - return the priority that is based on the static
> * priority but is modified by bonuses/penalties.
> *
> * We scale the actual sleep average [0 ... MAX_SLEEP_AVG]
> * We scale the actual sleep average [0 ... max_sleep_avg]

```

Sep 05, 06 23:43

code-review.txt

Page 16/22

```

> * into the -5 ... 0 ... +5 bonus/penalty range.
> *
> * We use 25% of the full 0...39 priority range so that:
@@ -714,7 +727,7 @@
> {
>     int bonus, prio;
>
> -     bonus = CURRENT_BONUS(p) - MAX_BONUS / 2;
> +     bonus = CURRENT_BONUS(p) - max_bonus / 2;
>
>     prio = p->static_prio - bonus;
>     if (prio < MAX_RT_PRIO)
@@ -738,7 +751,7 @@
> * If static_prio_timeslice() is ever changed to break this assumption then
> * this code will need modification
> */
> -#define TIME_SLICE_NICE_ZERO DEF_TIMESLICE
> +#define TIME_SLICE_NICE_ZERO def_timeslice
> #define LOAD_WEIGHT(lp) \
>     (((lp) * SCHED_LOAD_SCALE) / TIME_SLICE_NICE_ZERO)
> #define PRIO_TO_LOAD_WEIGHT(prio) \
@@ -911,8 +924,8 @@
>     p->sleep_avg += sleep_time;
>
>     }
> -     if (p->sleep_avg > NS_MAX_SLEEP_AVG)
> -         p->sleep_avg = NS_MAX_SLEEP_AVG;
> +     if (p->sleep_avg > ns_max_sleep_avg)
> +         p->sleep_avg = ns_max_sleep_avg;
> }
>
>     return effective_prio(p);
@@ -1620,7 +1633,7 @@
> * (current) is done further down, under its lock.
> */
> p->sleep_avg = JIFFIES_TO_NS(CURRENT_BONUS(p) *
>     CHILD_PENALTY / 100 * MAX_SLEEP_AVG / MAX_BONUS);
> +     child_penalty / 100 * max_sleep_avg / max_bonus);
>
>     p->prio = effective_prio(p);
@@ -1673,7 +1686,7 @@
>     this_rq = task_rq_lock(current, &flags);
> }
> current->sleep_avg = JIFFIES_TO_NS(CURRENT_BONUS(current) *
> -     PARENT_PENALTY / 100 * MAX_SLEEP_AVG / MAX_BONUS);
> +     parent_penalty / 100 * max_sleep_avg / max_bonus);
> task_rq_unlock(this_rq, &flags);
> }
>
@@ -1703,8 +1716,8 @@
> }
> if (p->sleep_avg < p->parent->sleep_avg)
>     p->parent->sleep_avg = p->parent->sleep_avg /
> -     (EXIT_WEIGHT + 1) * EXIT_WEIGHT + p->sleep_avg /
> -     (EXIT_WEIGHT + 1);
> +     (exit_weight + 1) * exit_weight + p->sleep_avg /
> +     (exit_weight + 1);
> task_rq_unlock(rq, &flags);
> }
>
@@ -2898,9 +2911,9 @@
> {
>     if (rq->curr->static_prio > rq->best_expired_prio)
>         return 1;
> -     if (!STARVATION_LIMIT || !rq->expired_timestamp)
> +     if (!starvation_limit || !rq->expired_timestamp)
>         return 0;
> -     if (jiffies - rq->expired_timestamp > STARVATION_LIMIT * rq->nr_running)
> +     if (jiffies - rq->expired_timestamp > starvation_limit * rq->nr_running)
>         return 1;
>     return 0;
> }
>
@@ -3183,8 +3196,8 @@
> * With real time tasks we run non-rt tasks only
> * per_cpu_gain% of the time.
> */
> -     if ((jiffies % DEF_TIMESLICE) >
> -         (sd->per_cpu_gain * DEF_TIMESLICE / 100))
> +     if ((jiffies % def_timeslice) >
> +         (sd->per_cpu_gain * def_timeslice / 100))
>         ret = 1;
>     } else {
>         if (smt_curr->static_prio < p->static_prio &&
@@ -3296,12 +3309,12 @@
>     schedstat_inc(rq, sched_cnt);
>     now = sched_clock();
> -     if (likely((long long)(now - prev->timestamp) < NS_MAX_SLEEP_AVG)) {
> +     if (likely((long long)(now - prev->timestamp) < ns_max_sleep_avg)) {
>         run_time = now - prev->timestamp;
>         if (unlikely((long long)(now - prev->timestamp) < 0))
>             run_time = 0;
>     } else
>         run_time = NS_MAX_SLEEP_AVG;
> +     run_time = ns_max_sleep_avg;
>
>     /*
> * Tasks charged proportionately less run_time at high sleep_avg to
@@ -3361,7 +3374,7 @@
>

```

Sep 05, 06 23:43

code-review.txt

Page 17/22

```

>
>         delta = 0;
>
>         if (next->sleep_type == SLEEP_INTERACTIVE)
> -             delta = delta * (ON_RUNQUEUE_WEIGHT * 128 / 100) / 128;
> +             delta = delta * (on_runqueue_weight * 128 / 100) / 128;
>
>         array = next->array;
>         new_prio = recal_task_prio(next, next->timestamp + delta);
@@ -6726,6 +6739,19 @@
> {
>     int i, j, k;
>
> +     min_timeslice = DEFAULT_MIN_TIMESLICE;
> +     def_timeslice = DEFAULT_DEF_TIMESLICE;
> +     on_runqueue_weight = DEFAULT_ON_RUNQUEUE_WEIGHT;
> +     child_penalty = DEFAULT_CHILD_PENALTY;
> +     parent_penalty = DEFAULT_PARENT_PENALTY;
> +     exit_weight = DEFAULT_EXIT_WEIGHT;
> +     prio_bonus_ratio = DEFAULT_PRIO_BONUS_RATIO;
> +     max_bonus = DEFAULT_MAX_BONUS;
> +     interactive_delta = DEFAULT_INTERACTIVE_DELTA;
> +     max_sleep_avg = DEFAULT_MAX_SLEEP_AVG;
> +     starvation_limit = DEFAULT_STARVATION_LIMIT;
> +     ns_max_sleep_avg = DEFAULT_NS_MAX_SLEEP_AVG;
>
>     for_each_possible_cpu(i) {
>         struct prio_array *array;
>         struct rq *rq;
>
> --
>
From moilanen@austin.vnet Tue Aug 1 16:41:31 2006
Subject: Re: [patch 8/8] scheduler: genetic library plugin
From: Jake Moilanen <moilanen@austin.vnet.ibm.com>
To: philips@vnet.ibm.com

On Mon, 2006-07-31 at 15:13 -0500, philips@vnet.ibm.com wrote:
> plain text document attachment (genetic-cpu)
> This patch attempts to tune the CPU scheduler using the genetic library. The
> current implementation is a proof of concept and under most workloads it shows
> a decrease in performance. The next steps will involve matching up statistics
> with genes in order to create a more fine tuned phenotype tree.

Tell me later how many of these nits I point out were in my code. :)

> Signed-off-by: Brandon Philips <philips@vnet.ibm.com>
> Index: linux-rc/kernel/sched.c
> =====
> --- linux-rc.orig/kernel/sched.c      2006-07-31 12:19:06.000000000 -0400
> +++ linux-rc/kernel/sched.c          2006-07-31 12:19:10.000000000 -0400
> @@ -16,6 +16,7 @@
> *
> *   by Davide Libenzi, preemptible kernel bits by Robert Love.
> *   2003-09-03   Interactivity tuning by Con Kolivas.
> *   2004-04-02   Scheduler domains code by Nick Piggin
> *   2006-06-28   Genetic library plugin by Brandon Philips
> */
>
> #include <linux/mm.h>
> @@ -56,6 +57,12 @@
>
> #include <asm/unistd.h>
>
> #ifdef CONFIG_GENETIC_CPU_SCHED
> #include <linux/genetic.h>
> #include <linux/random.h>
> #endif
>
> +
> /*
> * Convert user-nice values [ -20 ... 0 ... 19 ]
> * to static priority [ MAX_RT_PRIO..MAX_PRIO-1 ],
> @@ -113,6 +120,93 @@
> unsigned long starvation_limit;
> unsigned long ns_max_sleep_avg;
>
> #ifdef CONFIG_GENETIC_CPU_SCHED
> #define CPU_NUM_CHILDREN      8
> +
> #define CPU_THROUGHPUT_UID    1
> #define CPU_THROUGHPUT_NUM_GENES    0
> +
> #define CPU_LATENCY_UID      2
> #define CPU_LATENCY_NUM_GENES    0
> +
> #define CPU_CONTEXT_SWITCH_UID    4
> #define CPU_CONTEXT_SWITCH_NUM_GENES    0
> +
> #define CPU_GENERAL_UID      (CPU_THROUGHPUT_UID | CPU_LATENCY_UID | CPU_CONTEXT_SWITCH_UID)
> #define CPU_GENERAL_NUM_GENES    12
> +
> +struct cpu_genes {
> +    unsigned long child_penalty;
> +    unsigned long parent_penalty;
> +    unsigned long min_timeslice;
> +    unsigned long def_timeslice;
> +    unsigned long on_runqueue_weight;
> +    unsigned long exit_weight;
> +    unsigned long prio_bonus_ratio;
> +    unsigned long max_bonus;

```

Tuesday September 05, 2006

17/68

Sep 05, 06 23:43

code-review.txt

Page 18/22

```
> + unsigned long starvation_limit;
> + unsigned long ns_max_sleep_avg;
> + unsigned long max_sleep_avg;
> + unsigned long interactive_delta;
> +};
```

Was there no real-time genes?

```
> +gene_param_t cpu_gene_param[CPU_GENERAL_NUM_GENES] = {
> +     { "child_penalty",  DEFAULT_CHILD_PENALTY/3, DEFAULT_CHILD_PENALTY*3, DEFAULT_CHILD_PENALTY, 0 },
> +     { "parent_penalty",  DEFAULT_PARENT_PENALTY/3, DEFAULT_PARENT_PENALTY*3, DEFAULT_PARENT_PENALTY, 0 },
> +     { "min_timeslice",  1, 20, 10, 0 },
> +     { "def_timeslice",  DEFAULT_DEF_TIMESLICE/3, DEFAULT_DEF_TIMESLICE*3, DEFAULT_DEF_TIMESLICE, 0 },
> +     { "on_runqueue_weight",  DEFAULT_ON_RUNQUEUE_WEIGHT/3, DEFAULT_ON_RUNQUEUE_WEIGHT*3, DEFAULT_ON_RUNQUEUE_WEIGHT, 0 },
> +     { "exit_weight",  DEFAULT_EXIT_WEIGHT/3, DEFAULT_EXIT_WEIGHT*3, DEFAULT_EXIT_WEIGHT, 0 },
> +     { "prio_bonus_ratio",  DEFAULT_PRIO_BONUS_RATIO/3, DEFAULT_PRIO_BONUS_RATIO*3, DEFAULT_PRIO_BONUS_RATIO, 0 },
> +     { "max_bonus",  DEFAULT_MAX_BONUS/3, DEFAULT_MAX_BONUS*3, DEFAULT_MAX_BONUS, 0 },
> +     { "starvation_limit",  DEFAULT_STARVATION_LIMIT/3, DEFAULT_STARVATION_LIMIT*3, DEFAULT_STARVATION_LIMIT, 0 },
> +     { "max_sleep_avg",  DEFAULT_MAX_SLEEP_AVG/3, DEFAULT_MAX_SLEEP_AVG*3, DEFAULT_MAX_SLEEP_AVG, 0 },
> +     { "ns_max_sleep_avg",  DEFAULT_NS_MAX_SLEEP_AVG/2, (DEFAULT_NS_MAX_SLEEP_AVG/2) + DEFAULT_NS_MAX_SLEEP_AVG, DEFAULT_NS_MAX_SL
EEP_AVG, 0 },
> +     { "interactive_delta",  1, DEFAULT_INTERACTIVE_DELTA*3, DEFAULT_INTERACTIVE_DELTA, 0 },
> +};
```

I would print out the defaults of all these, and make sure that when you divide by 3, that the min doesn't go to 0. And if it does go to 0, individually audit them to make sure that's legal. (0 gives problems). I noticed that you did that w/ interactive_delta...

Also, many of these are probably going to need iterative mutations. I would be shocked if def_timeslice didn't need it.

```
> +static void cpu_take_stats_snapshot(phenotype_t * pt);
> +static void cpu_general_create_child(genetic_child_t * child);
> +static void cpu_general_set_child_genes(void * in_genes);
> +static void cpu_throughput_create_child(genetic_child_t * child);
> +static void cpu_latency_create_child(genetic_child_t * child);
> +static void cpu_context_switch_create_child(genetic_child_t * child);
> +static void cpu_throughput_calc_fitness(genetic_child_t * child);
> +static void cpu_latency_calc_fitness(genetic_child_t * child);
> +static void cpu_context_switch_calc_fitness(genetic_child_t * child);
> +static void cpu_general_calc_post_fitness(phenotype_t * in_pt);
> +static void cpu_shift_mutation_rate(phenotype_t * in_pt);
> +
> +struct sched_info * cpu_stats_snapshot;
> +
> +struct genetic_ops cpu_general_genetic_ops = {
> +     .create_child = cpu_general_create_child,          /* done */
> +     .set_child_genes = cpu_general_set_child_genes,   /* done */
> +     .combine_genes = genetic_generic_combine_genes,
> +     .mutate_child = genetic_generic_mutate_child,
> +     .calc_post_fitness = cpu_general_calc_post_fitness, /* done */
> +     .take_snapshot = cpu_take_stats_snapshot,        /* done */
> +     .shift_mutation_rate = cpu_shift_mutation_rate, /* done */
> +     .gene_show = genetic_generic_gene_show,
> +};
> +
> +struct genetic_ops cpu_throughput_genetic_ops = {
> +     .create_child = cpu_throughput_create_child,
> +     .calc_fitness = cpu_throughput_calc_fitness,
> +};
> +
> +struct genetic_ops cpu_latency_genetic_ops = {
> +     .create_child = cpu_latency_create_child,
> +     .calc_fitness = cpu_latency_calc_fitness,
> +};
> +
> +struct genetic_ops cpu_context_switch_genetic_ops = {
> +     .create_child = cpu_context_switch_create_child,
> +     .calc_fitness = cpu_context_switch_calc_fitness,
> +};
> +
> +#endif
> +
> +/*
> + * If a task is 'interactive' then we reinsert it in the active
> + * array after it has expired its current timeslice. (it will not
> + @@ -290,7 +384,7 @@
> + #define for_each_domain(cpu, __sd) \
> +     for (__sd = rcu_dereference(cpu_rq(cpu)->sd); __sd; __sd = __sd->parent)
> +
> + #define cpu_rq(cpu)          (&per_cpu(runqueues, (cpu)))
> + #define cpu_rq(cpu)          (&per_cpu(runqueues, (cpu)))
```

Was this a white space change? If so, take it out.

```
> #define this_rq()          (&__get_cpu_var(runqueues))
> #define task_rq(p)        cpu_rq(task_cpu(p))
> #define cpu_curr(cpu)     (cpu_rq(cpu)->curr)
> @@ -302,6 +396,8 @@
> # define finish_arch_switch(prev)    do { } while (0)
> #endif
> +
> +
```

ditto.

```
> #ifndef __ARCH_WANT_UNLOCKED_CTXSW
> static inline int task_running(struct rq *rq, struct task_struct *p)
```

Sep 05, 06 23:43

code-review.txt

Page 19/22

```

> {
> @@ -6900,3 +6996,242 @@
> }
>
> #endif
> +
> +#ifdef CONFIG_GENETIC_CPU_SCHED
> +static int genetic_cpu_sched_init(void)
> +{
> +    int ret;
> +    genetic_t * genetic = NULL;
> +
> +    cpu_stats_snapshot = (struct sched_info *)kmalloc(sizeof(struct sched_info), GFP_KERNEL);
> +
> +    if (!cpu_stats_snapshot)
> +        panic("sched_init: failed to malloc enough space");
> +
Looks like you forgot to change this functions prints to
"genetic_cpu_sched_init".
> +
> +    ret = genetic_init(&genetic, CPU_NUM_CHILDREN, 2 * HZ, 0, "ohone-cpuscheduler");
> +
This might be better at 3-4 seconds to start. I wasn't able to get
enough intelligible data at first w/ just 2 seconds.
> +
> +    if (ret)
> +        panic("sched_init: failed to init genetic lib");
> +
> +    if(genetic_register_phenotype(genetic, &cpu_throughput_genetic_ops, CPU_NUM_CHILDREN,
> +                                "throughput", CPU_THROUGHPUT_NUM_GENES, CPU_THROUGHPUT_UID))
> +        panic("sched_init: failed to register throughput phenotype");
> +
> +    if(genetic_register_phenotype(genetic, &cpu_latency_genetic_ops, CPU_NUM_CHILDREN,
> +                                "latency", CPU_LATENCY_NUM_GENES, CPU_LATENCY_UID))
> +        panic("sched_init: failed to register latency phenotype");
> +
> +    if(genetic_register_phenotype(genetic, &cpu_context_switch_genetic_ops, CPU_NUM_CHILDREN,
> +                                "context-switch", CPU_CONTEXT_SWITCH_NUM_GENES, CPU_CONTEXT_SWITCH_UID))
> +        panic("sched_init: failed to register context_switch phenotype");
> +
> +    if(genetic_register_phenotype(genetic, &cpu_general_genetic_ops, CPU_NUM_CHILDREN,
> +                                "general", CPU_GENERAL_NUM_GENES, CPU_GENERAL_UID))
> +        panic("sched_init: failed to register general phenotype");
> +
> +    genetic_start(genetic);
> +
> +    return 0;
> +}
> +postcore_initcall(genetic_cpu_sched_init);
> +
> +
> +#define child_stats(child) ((struct sched_info *) (child)->stats_snapshot)
> +
> +static void cpu_take_stats_snapshot(phenotype_t * pt)
> +{
> +    int cpu;
> +    struct sched_info * ss = child_stats(pt->child_ranking[0]);
> +
> +    memset(ss, 0, sizeof(struct sched_info));
> +
Add a comment about resetting count. (even if I didn't do it in my
code.) :)
> +
> +    for_each_online_cpu(cpu) {
> +        struct rq *rq = cpu_rq(cpu);
> +
> +        ss->cpu_time += rq->rq_sched_info.cpu_time;
> +        ss->run_delay += rq->rq_sched_info.run_delay;
> +        ss->pcnt += rq->rq_sched_info.pcnt;
> +    }
> +}
> +
> +static void cpu_general_create_child(genetic_child_t * child)
> +{
> +    BUG_ON(!child);
> +    child->genes = (void *)kmalloc(sizeof(struct cpu_genes), GFP_KERNEL);
> +    if (!child->genes)
> +        panic("cpu_general_create_child: error mallocing space");
> +
> +    child->num_genes = CPU_GENERAL_NUM_GENES;
> +    child->gene_param = cpu_gene_param;
> +    child->stats_snapshot = cpu_stats_snapshot;
> +
> +    genetic_create_child_defaults(child);
> +}
> +
> +static void cpu_general_set_child_genes(void * in_genes)
> +{
> +    struct cpu_genes * genes = (struct cpu_genes *)in_genes;
> +
> +    child_penalty = genes->child_penalty;
> +    parent_penalty = genes->parent_penalty;
> +    min_timeslice = genes->min_timeslice;
> +    def_timeslice = genes->def_timeslice;
> +
Add a check that def_timeslice isn't lower than min_timeslice. And if
it is, set min_timeslice to def_timeslice.
> +
> +    on_runqueue_weight = genes->on_runqueue_weight;

```

Tuesday September 05, 2006

19/68

Sep 05, 06 23:43

code-review.txt

Page 20/22

```

> +     exit_weight = genes->exit_weight;
> +     prio_bonus_ratio = genes->prio_bonus_ratio;
> +     max_bonus = genes->max_bonus;

Hmm...I might set this to (max_user_prio * prio_bonus_ratio / 100) for
now...I'd have to look at the code...but maybe make the gene the
divisor.

> +     starvation_limit = genes->starvation_limit;

I might set this to max_sleep_average.

> +     ns_max_sleep_avg = genes->ns_max_sleep_avg;

Start off w/ this equal to JIFFIES_TO_NS(max_sleep_average).

> +     interactive_delta = genes->interactive_delta;
> +     max_sleep_avg = genes->max_sleep_avg;

Start off w/ this as (def_timeslice * max_bonus).

> +}
> +
> +static void inline cpu_create_child(genetic_child_t * child)
> +{
> +     BUG_ON(!child);
> +
> +     child->genes = 0;
> +     child->gene_param = 0;
> +     child->num_genes = 0;
> +     child->stats_snapshot = cpu_stats_snapshot;
> +}
> +
> +static void cpu_throughput_create_child(genetic_child_t * child)
> +{
> +     cpu_create_child(child);
> +}
> +
> +static void cpu_latency_create_child(genetic_child_t * child)
> +{
> +     cpu_create_child(child);
> +}
> +
> +static void cpu_context_switch_create_child(genetic_child_t * child)
> +{
> +     cpu_create_child(child);
> +}
> +
> +
> +static void cpu_throughput_calc_fitness(genetic_child_t * child)
> +{
> +     int cpu;
> +     unsigned int cpu_time = 0;

Imagine a 64-way machine...Wont this overflow?

> +     struct sched_info * ss = child_stats(child);
> +
> +     for_each_online_cpu(cpu) {
> +         struct rq *rq = cpu_rq(cpu);
> +
> +         cpu_time += rq->rq_sched_info.cpu_time;
> +     }
> +
> +     child->fitness = cpu_time - ss->cpu_time;
> +}
> +
> +static void cpu_latency_calc_fitness(genetic_child_t * child)
> +{
> +     int cpu;
> +     unsigned int run_delay = 0;

ditto

> +     struct sched_info * ss = child_stats(child);
> +
> +     for_each_online_cpu(cpu) {
> +         struct rq *rq = cpu_rq(cpu);
> +
> +         run_delay += rq->rq_sched_info.run_delay;
> +     }
> +
> +     child->fitness = (run_delay - ss->run_delay);
> +     child->fitness = -child->fitness;
> +}
> +
> +static void cpu_context_switch_calc_fitness(genetic_child_t * child)
> +{
> +     int cpu;
> +     unsigned int pcnt = 0;

ditto

> +     struct sched_info * ss = child_stats(child);
> +
> +     for_each_online_cpu(cpu) {
> +         struct rq *rq = cpu_rq(cpu);
> +
> +         pcnt += rq->rq_sched_info.pcnt;
> +     }

```

Sep 05, 06 23:43

code-review.txt

Page 21/22

```

> +     child->fitness = (pcnt - ss->pcnt);
> +
I know we had this conversation...but pcnt was the number of context
switches??
> +     child->fitness = -child->fitness;
> +}
> +
> +static void cpu_general_calc_post_fitness(phenotype_t * in_pt)
> +{
> +     struct list_head * p;
> +     phenotype_t * pt;
> +     genetic_t * genetic = in_pt->genetic;
> +     int ranking[CPU_NUM_CHILDREN];
> +     int weight = 1;

No need to set this to 1. You unconditionally set it...or else BUG().

> +     int i;
> +
> +     memset(ranking, 0, sizeof(ranking));

Comment as to why zeroing out...

> +
> +     list_for_each(p, &genetic->phenotype) {
> +         pt = list_entry(p, phenotype_t, phenotype);
> +
> +         /* Look at everyone else that contributes to this
> +         phenotype */

Style:
/* Look at ...
 * phenotype
 */
> +         if (pt->uid & CPU_GENERAL_UID && pt->uid != CPU_GENERAL_UID) {
> +
> +             switch (pt->uid) {
> +                 case CPU_CONTEXT_SWITCH_UID:
> +                     weight = 2;

I would put less emphasis on this.

> +                     break;
> +                 case CPU_THROUGHPUT_UID:
> +                     weight = 2;
> +                     break;
> +                 case CPU_LATENCY_UID:
> +                     weight = 1;

And more on this.

> +                     break;
> +                 default:
> +                     BUG();
> +             }
> +
> +             for (i = 0; i < pt->num_children; i++)
> +                 ranking[pt->child_ranking[i]->id] += (i * weight);
> +         }
> +
> +         for (i = 0; i < in_pt->num_children; i++)
> +             in_pt->child_ranking[i]->fitness = ranking[i];
> +}
> +static void cpu_shift_mutation_rate(phenotype_t * in_pt)
> +{
> +     struct list_head * p;
> +     phenotype_t * pt;
> +     int count = 0;
> +     long rate = 0;
> +
> +     list_for_each(p, &in_pt->genetic->phenotype) {
> +         pt = list_entry(p, phenotype_t, phenotype);
> +
> +         /* Look at everyone else that contributes to this
> +         phenotype */
> +         if (pt->uid & CPU_GENERAL_UID && pt->uid != CPU_GENERAL_UID) {
> +
> +             switch (pt->uid) {
> +                 case CPU_CONTEXT_SWITCH_UID:
> +                 case CPU_THROUGHPUT_UID:
> +                 case CPU_LATENCY_UID:
> +                     rate += pt->mutation_rate;
> +                     count++;
> +                     break;
> +                 default:
> +                     BUG();
> +             }
> +         }
> +
> +         /* If we are a general phenotype that is made up of other
> +         phenotypes then we take the average */
> +         if (count)
> +             in_pt->mutation_rate = (rate / count);

```

```

> +     else
> +         BUG();
> +}
> +#endif
> Index: linux-rc/arch/i386/Kconfig
> =====
> --- linux-rc.orig/arch/i386/Kconfig      2006-07-31 12:14:45.000000000 -0400
> +++ linux-rc/arch/i386/Kconfig          2006-07-31 12:19:10.000000000 -0400
> @@ -258,6 +258,7 @@
>     increased overhead in some places. If unsure say N here.
>
>     source "kernel/Kconfig.preempt"
> +source "kernel/Kconfig.sched"
>
>     config X86_UP_APIC
>         bool "Local APIC support on uniprocessors"
> Index: linux-rc/arch/powerpc/Kconfig
> =====
> --- linux-rc.orig/arch/powerpc/Kconfig   2006-07-31 12:14:45.000000000 -0400
> +++ linux-rc/arch/powerpc/Kconfig        2006-07-31 12:19:10.000000000 -0400
> @@ -592,6 +592,7 @@
>
>     source kernel/Kconfig.hz
>     source kernel/Kconfig.preempt
> +source kernel/Kconfig.sched
>     source "fs/Kconfig.binfmt"
>
>     # We optimistically allocate largepages from the VM, so make the limit
> Index: linux-rc/arch/um/Kconfig
> =====
> --- linux-rc.orig/arch/um/Kconfig        2006-07-31 12:14:45.000000000 -0400
> +++ linux-rc/arch/um/Kconfig             2006-07-31 12:19:10.000000000 -0400
> @@ -107,6 +107,7 @@
>     <http://www.tldp.org/docs.html#howto>.
>
>
> +source "kernel/Kconfig.sched"
>     source "fs/Kconfig.binfmt"
>
>     config HOSTFS
> Index: linux-rc/kernel/Kconfig.sched
> =====
> --- /dev/null 1970-01-01 00:00:00.000000000 +0000
> +++ linux-rc/kernel/Kconfig.sched        2006-07-31 12:19:10.000000000 -0400
> @@ -0,0 +1,7 @@
> +config GENETIC_CPU_SCHED
> +     bool "Genetic CPU scheduler (EXPERIMENTAL)"
> +     depends on GENETIC_LIB && SCHEDSTATS && EXPERIMENTAL
> +     default n
> +     help
> +     This will tune the O(1) CPU Scheduler tunables dynamically based on
> +     workload statistics.

```

Can you go through all supported archs and turn on the necessary CONFIG options in the defconfigs...This makes it easy for "Joe Linux" to just apply, build and run...I'm guessing 95% of users don't know how to read the patch to realize they need SCHEDSTATS.

```
#!/bin/sh
# the next line restarts using tclsh \
exec wish "$@" "$@"

# genetk - graph the statistics of a genetic algorithm in real time
#
# Brandon Philips <brandon@ifup.org>
# Copyright (C) 2006 IBM
#
# This program is free software; you can redistribute it and/or modify it under
# the terms of version 2 of the GNU General Public License as published by the
# Free Software Foundation.
#
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
# details.
#
# You should have received a copy of the GNU General Public License along with
# this program; if not, write to the Free Software Foundation, Inc., 51
# Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

package require BLT
namespace import blt::*

# TODO generalize, take a parameter, parse stat file, etc
set prefix "/debug/genetic/"
set interval 2000
set interval_count 0
set window 20 ;# Number of ticks displayed at a time
set auto_scroll 1

vector create timeline ;# Vector of child timeslices, X vector for every graph

proc every {ms body} {eval $body; after $ms [info level 0]} ;# RS

# General purpose routine that parses out a "name: value" file into a list
proc parse_file {filename} {
    set file [open $filename r]

    while {[gets $file line] >= 0} {
        set raw [split $line ":"]
        set contents([string trim [lindex $raw 0]]\
                    [string trim [lindex $raw 1]])
    }
    close $file

    return [array get contents]
}

# Create a graph for each gene
#
# TODO Layout manager!
# relevant statistics

proc graph_set args {
    global select_state

    if {[info exists select_state("genes")]} {return}

    foreach name $select_state("genes") { eval .$name.g $args }
}

proc graph_scrollbar_command args {
    global auto_scroll interval_count window
    if {$interval_count > $window} {
        set auto_scroll 0
        eval graph_set $args
    }
}

proc graph_reset {} {
    graph_set axis configure x -min {} -max {}
}

proc graph_scroll args {
    eval .sb set $args
}

proc delete_gene_windows {} {
    global select_state

    if {[info exists select_state("genes")]} {return}

    foreach name $select_state("genes") {
        destroy .$name
    }
}

proc delete_gene_entries {genes} {
    $genes delete 0 end
}

proc add_selected_genes {src} {
    global select_state

    foreach index [$src curseselection] {
        set name [$src get $index]

        if {([array names select_state "*genes*" ] == "") || ([lsearch -exact $select_state("genes") $name] == -1)} {

```

```

        puts [array names select_state]
        lappend select_state("genes") $name
        create_gene_windows $name
    }
}

proc phenotype_path {} {
    global select_state prefix

    if {[info exists select_state("phenotype")]} {
        return "$prefix/$select_state("plugin")/phenotypes/$select_state("phenotype")"
    }

    return -1
}

proc add_genes_to_listbox {src dst} {
    global select_state prefix
    delete_gene_windows
    delete_gene_entries $dst
    unset -nocomplain select_state("phenotype") select_state("genes")

    set select_state("phenotype") [$src get [$src curselection]]

    array set contents [parse_file [phenotype_path]]
    foreach name [array names contents] {
        $dst insert end $name
    }
}

proc add_phenotypes_to_listbox {src dst genes} {
    global select_state prefix
    delete_gene_windows
    delete_gene_entries $genes
    unset -nocomplain select_state("phenotype") select_state("genes")

    $dst delete 0 end
    set select_state("plugin") [$src get [$src curselection]]
    set namepat "$prefix$select_state("plugin")/phenotypes/*"
    set phenotype [glob -types f -- $namepat]
    $dst insert end [file tail $phenotype]
}

proc gene_select { parent available_plugins } {
    frame $parent
    set plugins [listbox $parent.plugins -width 20 -height 10 -selectmode single]
    set phenotypes [listbox $parent.phenotypes -width 20 -height 10 -selectmode single]
    set genes [listbox $parent.genes -width 20 -height 10 -selectmode extended]

    pack $plugins $phenotypes $genes -side left

    bind $plugins <ButtonRelease-1> \
        [list add_phenotypes_to_listbox %W $phenotypes $genes]

    bind $phenotypes <ButtonRelease-1> \
        [list add_genes_to_listbox %W $genes]

    bind $genes <ButtonRelease-1> \
        [list add_selected_genes %W]

    foreach x $available_plugins {
        $plugins insert end [file tail $x]
    }
}

proc create_gene_windows genes {
    puts $genes
    foreach name $genes {
        if {[vector names $name] != ""} {continue}
        vector create $name

        toplevel .$name
        graph .$name.g

        .$name.g axis configure -scrollcommand graph_scroll
        .$name.g element create $name -symbol "scross" -label "" -xdata timeline -ydata $name

        pack .$name.g
    }
}

proc get_filename {} {
    global select_state
    if {[info exists $select_state("plugin")]
        && [info exists $select_state("phenotype")]
        && [info exists $select_state("genes")]} {
        return
    }
    return 0
}

proc update_graphs {} {
    global interval_count interval window auto_scroll select_state
    timeline variable timeline
    incr interval_count

```

Sep 05, 06 0:09

genetk

Page 3/3

```

array set contents [parse_file [phenotype_path]]

foreach key $select_state("genes") {
    $key variable vec
    set vec(++end) $contents($key)
}

if {$auto_scroll != 0} {
    graph_set axis configure x -min [expr $interval * ($interval_count - $window)] -max [expr $interval * $interval_count]
}

set next [expr $timeline(max) + $interval]
set timeline(++end) $next
}

# Scrollbar for moving the data sets left and right in time
pack [scrollbar .sb -command [list graph_scrollbar_command axis view x] -orient horizontal] -fill both -expand true
# Checkbox for the auto scroll
pack [checkboxbutton .reset -variable auto_scroll -text "Auto Scroll"] -fill both -expand true

gene_select .glist [glob "$prefix*" -types d]
pack .glist

# Collect and display gene values every $child_lifetime seconds
#
# TODO get runtime of children from debugfs
# update the min and max X value to display
every $interval {
    global select_state

    if {[phenotype_path] != -1} && ([info exists select_state("genes")]) { update_graphs }
}

```

Sep 05, 06 0:16

genetic-as-sched.patch

Page 1/5

```
block: Anticipatory I/O scheduler genetic plugin
```

This plugin shows a worthwhile performance increase with most workloads averaging an 8% improvement. The implementation is fairly straightforward with the modification of the parameters `as_general_set_child_genes`.

```
Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>
```

```
---
block/Kconfig.iosched |      9 +
block/as-iosched.c    | 353 ++++++-----
2 files changed, 361 insertions(+), 1 deletion(-)
```

```
Index: linux-gl/block/Kconfig.iosched
```

```
-----
--- linux-gl.orig/block/Kconfig.iosched
+++ linux-gl/block/Kconfig.iosched
@@ -66,4 +66,13 @@ config DEFAULT_IOSCHED
     default "cfq" if DEFAULT_CFQ
     default "noop" if DEFAULT_NOOP
```

```
+config GENETIC_IOSCHED_AS
+    bool "Genetic Anticipatory I/O scheduler (EXPERIMENTAL)"
+    depends on IOSCHED_AS && GENETIC_LIB && EXPERIMENTAL
+    default n
+    ---help---
+    This will use a genetic algorithm to tweak the tunables of the
+    anticipatory scheduler automatically and will adapt tunables
+    depending on the present workload.
```

```
+endmenu
```

```
Index: linux-gl/block/as-iosched.c
```

```
-----
--- linux-gl.orig/block/as-iosched.c
+++ linux-gl/block/as-iosched.c
@@ -17,6 +17,9 @@
#include <linux/hash.h>
#include <linux/rbtree.h>
#include <linux/interrupt.h>
+#include <linux/genetic.h>
+#include <linux/random.h>
+#include <linux/debugfs.h>
```

```
#define REQ_SYNC      1
#define REQ_ASYNC     0
@@ -64,6 +67,8 @@
*/
#define MAX_THINKTIME (HZ/50UL)
```

```
+unsigned long max_thinktime = MAX_THINKTIME;
+
+/* Bits in as_io_context.state */
+enum as_io_states {
+    AS_TASK_RUNNING=0, /* Process has not exited */
+};
@@ -80,6 +85,109 @@ enum anticipation_status {
    * or timed out */
};
```

```
+#ifdef CONFIG_GENETIC_IOSCHED_AS
```

```
+
+struct disk_stats_snapshot * as_stats_snapshot;
+
+extern void disk_stats_snapshot(phenotype_t * pt);
+#ifdef CONFIG_FINGERPRINTING
+extern void disk_get_fingerprint(phenotype_t * pt);
+extern void disk_update_fingerprint(phenotype_t * pt);
+extern void * as_create_genes(phenotype_t * pt);
+#endif
+
+static void as_num_ops_create_child(genetic_child_t * child);
+static void as_throughput_create_child(genetic_child_t * child);
+static void as_latency_create_child(genetic_child_t * child);
+static void as_general_create_child(genetic_child_t * child);
+
+static void as_general_set_child_genes(void * in_genes);
+
+static void as_num_ops_calc_fitness(genetic_child_t * child);
+static void as_throughput_calc_fitness(genetic_child_t * child);
+static void as_latency_calc_fitness(genetic_child_t * child);
+
+static void as_general_calc_post_fitness(phenotype_t * in_pt);
+
+static void as_shift_mutation_rate(phenotype_t * in_pt);
+
+struct genetic_ops as_num_ops_genetic_ops = {
+    .create_child = as_num_ops_create_child,
+    .calc_fitness = as_num_ops_calc_fitness,
+};
+
+struct genetic_ops as_throughput_genetic_ops = {
+    .create_child = as_throughput_create_child,
+    .calc_fitness = as_throughput_calc_fitness,
+};
+
+struct genetic_ops as_latency_genetic_ops = {
+    .create_child = as_latency_create_child,
+    .calc_fitness = as_latency_calc_fitness,
+};
+
+struct genetic_ops as_general_genetic_ops = {
+    .create_child = as_general_create_child,
```

Sep 05, 06 0:16

genetic-as-sched.patch

Page 2/5

```

+ .set_child_genes = as_general_set_child_genes,
+ .combine_genes = genetic_generic_combine_genes,
+ .mutate_child = genetic_generic_mutate_child,
+ .calc_post_fitness = as_general_calc_post_fitness,
+ .take_snapshot = disk_stats_snapshot,
+ .shift_mutation_rate = as_shift_mutation_rate,
+#ifdef CONFIG_FINGERPRINTING
+ .get_fingerprint = disk_get_fingerprint,
+ .update_fingerprint = disk_update_fingerprint,
+ .create_top_genes = as_create_genes,
+ .top_fitness_show = fingerprint_top_fitness_show,
+ .snapshot_show = fingerprint_snapshot_show,
+ .state_show = fingerprint_state_show,
+#endif
+};
+
+#define AS_NUM_CHILDREN 8
+
+#define AS_NUM_OPS_UID 1
+#define AS_NUM_OPS_NUM_GENES 0
+
+#define AS_THROUGHPUT_UID 2
+#define AS_THROUGHPUT_NUM_GENES 0
+
+#define AS_LATENCY_UID 4
+#define AS_LATENCY_NUM_GENES 0
+
+#define AS_GENERAL_UID (AS_NUM_OPS_UID | AS_THROUGHPUT_UID | AS_LATENCY_UID)
+#define AS_GENERAL_NUM_GENES 7
+
+struct as_genes {
+ unsigned long read_expire;
+ unsigned long write_expire;
+ unsigned long read_batch_expire;
+ unsigned long write_batch_expire;
+ unsigned long antic_expire;
+ unsigned long max_thinktime;
+ unsigned long nr_requests;
+};
+
+#define AS_TUNABLE(var, name) GENETIC_TUNABLE(var, name)
+
+gene_param_t as_gene_param[AS_GENERAL_NUM_GENES] = {
+ {"read_expire", HZ/16, 3*HZ/16, default_read_expire, 0},
+ {"write_expire", HZ/8, 3*HZ/8, default_write_expire, 0},
+ {"read_batch_expire", HZ/4, 3*HZ/4, default_read_batch_expire, 0},
+ {"write_batch_expire", HZ/16, 3*HZ/16, default_write_batch_expire, 0},
+ // {"default_antic_expire", HZ/300, HZ/100, default_antic_expire, 0},
+ {"default_antic_expire", 0, HZ/100, default_antic_expire, 0},
+ {"max_thinktime", HZ/100, 3*HZ/100, MAX_THINKTIME, 0},
+ {"nr_requests", BLKDEV_MIN_RQ, BLKDEV_MAX_RQ*30, BLKDEV_MAX_RQ, genetic_generic_iterative_mutate_gene}
+};
+
+extern long long disk_num_ops_calc_fitness(genetic_child_t * child);
+extern long long disk_throughput_calc_fitness(genetic_child_t * child);
+extern long long disk_latency_calc_fitness(genetic_child_t * child);
+
+LIST_HEAD(as_data_list);
+#endif
+
+struct as_data {
+ /*
+  * run time data
+  */
+@@ -131,6 +239,9 @@ struct as_data {
+ unsigned long fifo_expire[2];
+ unsigned long batch_expire[2];
+ unsigned long antic_expire;
+#ifdef CONFIG_GENETIC_IOSCHED_AS
+ struct list_head data_list;
+#endif
+};
+
+#define list_entry_fifo(ptr) list_entry(ptr, struct as_rq, fifo)
+@@ -730,7 +841,7 @@ static void as_update_ichist(struct as_d
+ if (test_bit(AS_TASK_IORUNNING, &aic->state)
+     && in_flight == 0) {
+ thinktime = jiffies - aic->last_end_request;
+ thinktime = min(thinktime, MAX_THINKTIME-1);
+ thinktime = min(thinktime, max_thinktime-1);
+ }
+ as_update_thinktime(ad, aic, thinktime);
+
+@@ -1628,6 +1739,11 @@ static void as_exit_queue(elevator_t *e)
+ mempool_destroy(ad->arq_pool);
+ put_io_context(ad->io_context);
+
+#ifdef CONFIG_GENETIC_IOSCHED_AS
+ list_del(&ad->data_list);
+#endif
+
+ kfree(ad->hash);
+ kfree(ad);
+ }
+@@ -1690,6 +1806,10 @@ static void *as_init_queue(request_queue
+ if (ad->write_batch_count < 2)
+ ad->write_batch_count = 2;
+
+#ifdef CONFIG_GENETIC_IOSCHED_AS
+ list_add_tail(&ad->data_list, &as_data_list);

```

```

#endif
+
+   return ad;
+
+ }

@@ -1805,6 +1925,9 @@ static struct elevator_type iosched_as =
static int __init as_init(void)
{
    int ret;
#ifdef CONFIG_GENETIC_IOSCHED_AS
+   genetic_t * genetic = 0;
#endif

    arq_pool = kmem_cache_create("as_arq", sizeof(struct as_rq),
                                0, 0, NULL, NULL);
@@ -1813,6 +1936,36 @@ static int __init as_init(void)

    ret = elv_register(&iosched_as);
    if (!ret) {
+
+ #ifdef CONFIG_GENETIC_IOSCHED_AS
+     as_stats_snapshot = (struct disk_stats_snapshot *)kmalloc(sizeof(struct disk_stats_snapshot), GFP_KERNEL);
+     if (!as_stats_snapshot)
+         panic("as: failed to malloc enough space");
+
+     ret = genetic_init(&genetic, AS_NUM_CHILDREN, 2 * HZ, 1, "as-ioscheduler");
+     if (ret)
+         panic("as: failed to init genetic lib");
+
+     if(genetic_register_phenotype(genetic, &as_num_ops_genetic_ops, AS_NUM_CHILDREN,
+                                   "num_ops", AS_NUM_OPS_NUM_GENES, AS_NUM_OPS_UID) == NULL)
+         panic("as: failed to register num_ops phenotype");
+
+     if(genetic_register_phenotype(genetic, &as_throughput_genetic_ops, AS_NUM_CHILDREN,
+                                   "throughput", AS_THROUGHPUT_NUM_GENES, AS_THROUGHPUT_UID) == NULL)
+         panic("as: failed to register throughput phenotype");
+
+     if(genetic_register_phenotype(genetic, &as_latency_genetic_ops, AS_NUM_CHILDREN,
+                                   "latency", AS_LATENCY_NUM_GENES, AS_LATENCY_UID) == NULL)
+         panic("as: failed to register latency phenotype");
+
+     if(genetic_register_phenotype(genetic, &as_general_genetic_ops, AS_NUM_CHILDREN,
+                                   "general", AS_GENERAL_NUM_GENES, AS_GENERAL_UID) == NULL)
+         panic("as: failed to register general phenotype");
+
+     genetic_start(genetic);
+ #endif
+
+     /*
+      * don't allow AS to get unregistered, since we would have
+      * to browse all tasks in the system and release their
    @@ -1839,6 +1992,204 @@ static void __exit as_exit(void)
        kmem_cache_destroy(arq_pool);
    }

#ifdef CONFIG_GENETIC_IOSCHED_AS
+
+static void as_num_ops_create_child(genetic_child_t * child)
+{
+   BUG_ON(!child);
+
+   child->genes = 0;
+   child->gene_param = 0;
+   child->num_genes = AS_NUM_OPS_NUM_GENES;
+   child->stats_snapshot = as_stats_snapshot;
+}
+
+static void as_throughput_create_child(genetic_child_t * child)
+{
+   BUG_ON(!child);
+
+   child->genes = 0;
+   child->gene_param = 0;
+   child->num_genes = AS_THROUGHPUT_NUM_GENES;
+   child->stats_snapshot = as_stats_snapshot;
+}
+
+static void as_latency_create_child(genetic_child_t * child)
+{
+   BUG_ON(!child);
+
+   child->genes = 0;
+   child->gene_param = 0;
+   child->num_genes = AS_LATENCY_NUM_GENES;
+   child->stats_snapshot = as_stats_snapshot;
+}
+
+/* need to create the genes for the child */
+static void as_general_create_child(genetic_child_t * child)
+{
+   BUG_ON(!child);
+
+   child->genes = (void *)kmalloc(sizeof(struct as_genes), GFP_KERNEL);
+   if (!child->genes)
+       panic("as_general_create_child: error mallocing space");
+
+   child->gene_param = as_gene_param;
+   child->num_genes = AS_GENERAL_NUM_GENES;
+   child->stats_snapshot = as_stats_snapshot;

```

```

+
+   genetic_create_child_spread(child, AS_NUM_CHILDREN-1);
+
+   ((struct as_genes *)child->genes)->nr_requests = BLKDEV_MAX_RQ;
+}
+
+static void as_shift_mutation_rate(phenotype_t * in_pt)
+{
+   struct list_head * p;
+   genetic_t * genetic = to_phenotype_genetic(in_pt);
+   phenotype_t * pt;
+   int count = 0;
+   long rate = 0;
+
+   list_for_each(p, &genetic->phenotypes.list) {
+       pt = to_phenotype(to_kobj(p));
+
+       /* Look at everyone else that contributes to this
+        * phenotype */
+       if (pt->uid & AS_GENERAL_UID && pt->uid != AS_GENERAL_UID) {
+
+           switch (pt->uid) {
+               case AS_NUM_OPS_UID:
+               case AS_THROUGHPUT_UID:
+               case AS_LATENCY_UID:
+                   rate += pt->mutation_rate;
+                   count++;
+                   break;
+               default:
+                   BUG();
+           }
+       }
+   }
+
+   /* If we are a general phenotype that is made up of other
+    * phenotypes then we take the average */
+   if (count)
+       in_pt->mutation_rate = (rate / count);
+   else
+       BUG();
+}
+
+static void as_general_set_child_genes(void * in_genes)
+{
+   struct as_genes * genes = (struct as_genes *)in_genes;
+   struct list_head * d;
+   struct as_data * ad;
+
+   list_for_each(d, &as_data_list) {
+       ad = list_entry(d, struct as_data, data_list);
+       ad->fifo_expire[REQ_SYNC] = genes->read_expire;
+       ad->fifo_expire[REQ_ASYNC] = genes->write_expire;
+       ad->antic_expire = genes->antic_expire;
+
+       if (genes->read_batch_expire > genes->write_expire)
+           genes->read_batch_expire = genes->write_expire;
+       ad->batch_expire[REQ_SYNC] = genes->read_batch_expire;
+
+       if (genes->write_batch_expire > genes->read_expire)
+           genes->write_batch_expire = genes->read_expire;
+       ad->batch_expire[REQ_ASYNC] = genes->write_batch_expire;
+
+       ad->q->nr_requests = genes->nr_requests;
+   }
+   max_thinktime = genes->max_thinktime;
+}
+
+static void as_num_ops_calc_fitness(genetic_child_t * child)
+{
+   child->fitness = disk_num_ops_calc_fitness(child);
+}
+
+static void as_throughput_calc_fitness(genetic_child_t * child)
+{
+   child->fitness = disk_throughput_calc_fitness(child);
+}
+
+static void as_latency_calc_fitness(genetic_child_t * child)
+{
+   child->fitness = disk_latency_calc_fitness(child);
+}
+
+/* Make the general the one that takes into account all the fitness
+ * routines, since these are the common genes that effect everything.
+ */
+static void as_general_calc_post_fitness(phenotype_t * in_pt)
+{
+   struct list_head * p;
+   phenotype_t * pt;
+   genetic_t * genetic = to_phenotype_genetic(in_pt);
+   int ranking[AS_NUM_CHILDREN];
+   int weight = 1;
+   int i;
+
+   memset(ranking, 0, sizeof(ranking));
+
+   list_for_each(p, &genetic->phenotypes.list) {
+       pt = to_phenotype(to_kobj(p));
+

```

```

+      /* Look at everyone else that contributes to this
+       phenotype */
+      if (pt->uid & AS_GENERAL_UID && pt->uid != AS_GENERAL_UID) {
+
+          switch (pt->uid) {
+          case AS_NUM_OPS_UID:
+              weight = 2;
+              break;
+          case AS_THROUGHPUT_UID:
+              weight = 2;
+              break;
+          case AS_LATENCY_UID:
+              weight = 1;
+              break;
+          default:
+              BUG();
+          }
+
+          for (i = 0; i < pt->num_children; i++)
+              ranking[pt->child_ranking[i]->id] += (i * weight);
+      }
+  }
+
+  for (i = 0; i < in_pt->num_children; i++)
+      in_pt->child_ranking[i]->fitness = ranking[i];
+}
+
+#ifdef CONFIG_FINGERPRINTING
+void * as_create_genes(phenotype_t * pt)
+{
+  struct as_genes * genes = (void *)kmalloc(sizeof(struct as_genes), GFP_KERNEL);
+  if (!genes) {
+      printk(KERN_ERR "as_create_genes: unable to alloc space\n");
+      return 0;
+  }
+
+  /* at some point...make these intelligent depending on what
+   * the workload is
+   */
+  genes->read_expire = default_read_expire;
+  genes->write_expire = default_write_expire;
+  genes->read_batch_expire = default_read_batch_expire;
+  genes->write_batch_expire = default_write_batch_expire;
+  genes->antic_expire = default_antic_expire;
+  genes->max_thinktime = MAX_THINKTIME;
+  genes->nr_requests = BLKDEV_MAX_RQ;
+
+  return (void *)genes;
+}
+#endif /* CONFIG_FINGERPRINTING */
+
+#endif
+
+module_init(as_init);
+module_exit(as_exit);

```

Sep 05, 06 0:16

genetic-cpu-constants.patch

Page 1/3

scheduler: Make tuning knobs global variables

This patch replaces all #define "tuning knobs" in the scheduler with global variables that can be later manipulated by a genetic library plugin for the scheduler.

Signed-off-by: Brandon Philips <philips@vnet.ibm.com>

```
---
kernel/sched.c | 106 ++++++-----
1 file changed, 66 insertions(+), 40 deletions(-)
```

Index: linux-gf/kernel/sched.c

```
=====
--- linux-gf.orig/kernel/sched.c
+++ linux-gf/kernel/sched.c
@@ -87,18 +87,31 @@
 * default timeslice is 100 msecs, maximum timeslice is 800 msecs.
 * Timeslices get refilled after they expire.
 */
-#define MIN_TIMESLICE          max(5 * HZ / 1000, 1)
-#define DEF_TIMESLICE          (100 * HZ / 1000)
-#define ON_RUNQUEUE_WEIGHT    30
-#define CHILD_PENALTY         95
-#define PARENT_PENALTY        100
-#define EXIT_WEIGHT           3
-#define PRIO_BONUS_RATIO       25
-#define MAX_BONUS              (MAX_USER_PRIO * PRIO_BONUS_RATIO / 100)
-#define INTERACTIVE_DELTA      2
-#define MAX_SLEEP_AVG          (DEF_TIMESLICE * MAX_BONUS)
-#define STARVATION_LIMIT      (MAX_SLEEP_AVG)
-#define NS_MAX_SLEEP_AVG      (JIFFIES_TO_NS(MAX_SLEEP_AVG))
+#define DEFAULT_MIN_TIMESLICE  max(5 * HZ / 1000, 1)
+#define DEFAULT_DEF_TIMESLICE  (100 * HZ / 1000)
+#define DEFAULT_ON_RUNQUEUE_WEIGHT 30
+#define DEFAULT_CHILD_PENALTY  95
+#define DEFAULT_PARENT_PENALTY 100
+#define DEFAULT_EXIT_WEIGHT     3
+#define DEFAULT_PRIO_BONUS_RATIO 25
+#define DEFAULT_MAX_BONUS       (MAX_USER_PRIO * DEFAULT_PRIO_BONUS_RATIO / 100)
+#define DEFAULT_INTERACTIVE_DELTA 2
+#define DEFAULT_MAX_SLEEP_AVG   (DEFAULT_DEF_TIMESLICE * DEFAULT_MAX_BONUS)
+#define DEFAULT_STARVATION_LIMIT (DEFAULT_MAX_SLEEP_AVG)
+#define DEFAULT_NS_MAX_SLEEP_AVG (JIFFIES_TO_NS(DEFAULT_MAX_SLEEP_AVG))
+
+unsigned long min_timeslice;
+unsigned long def_timeslice;
+unsigned long on_runqueue_weight;
+unsigned long child_penalty;
+unsigned long parent_penalty;
+unsigned long exit_weight;
+unsigned long prio_bonus_ratio;
+unsigned long max_bonus;
+unsigned long interactive_delta;
+unsigned long max_sleep_avg;
+unsigned long starvation_limit;
+unsigned long ns_max_sleep_avg;
+
+/*
+ * If a task is 'interactive' then we reinsert it in the active
+ */
@@ -108,7 +121,7 @@
 *
 * This part scales the interactivity limit depending on niceness.
 *
- * We scale it linearly, offset by the INTERACTIVE_DELTA delta.
+ * We scale it linearly, offset by the interactive_delta delta.
 * Here are a few examples of different nice levels:
 *
 * TASK_INTERACTIVE(-20): [1,1,1,1,1,1,1,1,0,0]
@@ -129,33 +142,33 @@
 */
-#define CURRENT_BONUS(p) \
- (NS_TO_JIFFIES((p)->sleep_avg) * MAX_BONUS / \
-  MAX_SLEEP_AVG)
+#define CURRENT_BONUS(p) \
+ (NS_TO_JIFFIES((p)->sleep_avg) * max_bonus / \
+  max_sleep_avg)
+
+#define GRANULARITY (10 * HZ / 1000 ? : 1)
+
+#ifdef CONFIG_SMP
+#define TIMESLICE_GRANULARITY(p) (GRANULARITY * \
+ (1 << (((MAX_BONUS - CURRENT_BONUS(p)) ? : 1) - 1))) * \
+ (1 << (((max_bonus - CURRENT_BONUS(p)) ? : 1) - 1))) * \
+ num_online_cpus())
+#else
+#define TIMESLICE_GRANULARITY(p) (GRANULARITY * \
+ (1 << (((MAX_BONUS - CURRENT_BONUS(p)) ? : 1) - 1)))
+ (1 << (((max_bonus - CURRENT_BONUS(p)) ? : 1) - 1)))
+#endif
+
+#define SCALE(v1,v1_max,v2_max) \
+ (v1) * (v2_max) / (v1_max)
+
+#define DELTA(p) \
+ (SCALE(TASK_NICE(p) + 20, 40, MAX_BONUS) - 20 * MAX_BONUS / 40 + \
+  INTERACTIVE_DELTA)
+ (SCALE(TASK_NICE(p) + 20, 40, max_bonus) - 20 * max_bonus / 40 + \
+  interactive_delta)
+
+#define TASK_INTERACTIVE(p) \
```

```

((p)->prio <= (p)->static_prio - DELTA(p))

#define INTERACTIVE_SLEEP(p) \
- (JIFFIES_TO_NS(MAX_SLEEP_AVG * \
+ (MAX_BONUS / 2 + DELTA((p)) + 1) / MAX_BONUS - 1))
+ (JIFFIES_TO_NS(max_sleep_avg * \
+ (max_bonus / 2 + DELTA((p)) + 1) / max_bonus - 1))

#define TASK_PREEMPTS_CURR(p, rq) \
((p)->prio < (rq)->curr->prio)
@@ -170,14 +183,14 @@
*/

#define SCALE_PRIO(x, prio) \
- max(x * (MAX_PRIO - prio) / (MAX_USER_PRIO / 2), MIN_TIMESLICE)
+ max(x * (MAX_PRIO - prio) / (MAX_USER_PRIO / 2), min_timeslice)

static unsigned int static_prio_timeslice(int static_prio)
{
- if (static_prio < NICE_TO_PRIO(0))
+ return SCALE_PRIO(DEF_TIMESLICE * 4, static_prio);
+ return SCALE_PRIO(def_timeslice * 4, static_prio);
- else
+ return SCALE_PRIO(DEF_TIMESLICE, static_prio);
+ return SCALE_PRIO(def_timeslice, static_prio);
}

static inline unsigned int task_timeslice(struct task_struct *p)
@@ -699,7 +712,7 @@ enqueue_task_head(struct task_struct *p,
* __normal_prio - return the priority that is based on the static
* priority but is modified by bonuses/penalties.
*
- * We scale the actual sleep average [0 ... MAX_SLEEP_AVG]
+ * We scale the actual sleep average [0 ... max_sleep_avg]
* into the -5 ... 0 ... +5 bonus/penalty range.
*
* We use 25% of the full 0...39 priority range so that:
@@ -714,7 +727,7 @@ static inline int __normal_prio(struct t
{
int bonus, prio;

- bonus = CURRENT_BONUS(p) - MAX_BONUS / 2;
+ bonus = CURRENT_BONUS(p) - max_bonus / 2;

prio = p->static_prio - bonus;
if (prio < MAX_RT_PRIO)
@@ -738,7 +751,7 @@ static inline int __normal_prio(struct t
* If static_prio_timeslice() is ever changed to break this assumption then
* this code will need modification
*/
-#define TIME_SLICE_NICE_ZERO DEF_TIMESLICE
+#define TIME_SLICE_NICE_ZERO def_timeslice
#define LOAD_WEIGHT(lp) \
(((lp) * SCHED_LOAD_SCALE) / TIME_SLICE_NICE_ZERO)
#define PRIO_TO_LOAD_WEIGHT(prio) \
@@ -911,8 +924,8 @@ static int recalc_task_prio(struct task_
p->sleep_avg += sleep_time;

}

- if (p->sleep_avg > NS_MAX_SLEEP_AVG)
+ p->sleep_avg = NS_MAX_SLEEP_AVG;
+ if (p->sleep_avg > ns_max_sleep_avg)
+ p->sleep_avg = ns_max_sleep_avg;
}

return effective_prio(p);
@@ -1620,7 +1633,7 @@ void fastcall wake_up_new_task(struct ta
* (current) is done further down, under its lock.
*/
p->sleep_avg = JIFFIES_TO_NS(CURRENT_BONUS(p) *
- CHILD_PENALTY / 100 * MAX_SLEEP_AVG / MAX_BONUS);
+ child_penalty / 100 * max_sleep_avg / max_bonus);

p->prio = effective_prio(p);

@@ -1673,7 +1686,7 @@ void fastcall wake_up_new_task(struct ta
this_rq = task_rq_lock(current, &flags);

current->sleep_avg = JIFFIES_TO_NS(CURRENT_BONUS(current) *
- PARENT_PENALTY / 100 * MAX_SLEEP_AVG / MAX_BONUS);
+ parent_penalty / 100 * max_sleep_avg / max_bonus);
task_rq_unlock(this_rq, &flags);
}

@@ -1703,8 +1716,8 @@ void fastcall sched_exit(struct task_str
}
if (p->sleep_avg < p->parent->sleep_avg)
p->parent->sleep_avg = p->parent->sleep_avg /
- (EXIT_WEIGHT + 1) * EXIT_WEIGHT + p->sleep_avg /
- (EXIT_WEIGHT + 1);
+ (exit_weight + 1) * exit_weight + p->sleep_avg /
+ (exit_weight + 1);
task_rq_unlock(rq, &flags);
}

@@ -2898,9 +2911,9 @@ static inline int expired_starving(struc
{
if (rq->curr->static_prio > rq->best_expired_prio)
return 1;
- if (!STARVATION_LIMIT || !rq->expired_timestamp)

```

Sep 05, 06 0:16

genetic-cpu-constants.patch

Page 3/3

```

+   if (!starvation_limit || !rq->expired_timestamp)
+       return 0;
-   if (jiffies - rq->expired_timestamp > STARVATION_LIMIT * rq->nr_running)
+   if (jiffies - rq->expired_timestamp > starvation_limit * rq->nr_running)
+       return 1;
+       return 0;
}
@@ -3183,8 +3196,8 @@ dependent_sleeper(int this_cpu, struct r
+       * With real time tasks we run non-rt tasks only
+       * per_cpu_gain% of the time.
+       */
-       if ((jiffies % DEF_TIMESLICE) >
+       (sd->per_cpu_gain * DEF_TIMESLICE / 100))
+       if ((jiffies % def_timeslice) >
+       (sd->per_cpu_gain * def_timeslice / 100))
+           ret = 1;
+       } else {
+       if (smt_curr->static_prio < p->static_prio &&
@@ -3296,12 +3309,12 @@ need_resched_nonpreemptible:
+       schedstat_inc(rq, sched_cnt);
+       now = sched_clock();
-       if (likely((long long)(now - prev->timestamp) < NS_MAX_SLEEP_AVG)) {
+       if (likely((long long)(now - prev->timestamp) < ns_max_sleep_avg)) {
+           run_time = now - prev->timestamp;
+           if (unlikely((long long)(now - prev->timestamp) < 0))
+               run_time = 0;
+       } else
+           run_time = NS_MAX_SLEEP_AVG;
+       run_time = ns_max_sleep_avg;
+
+       /*
+       * Tasks charged proportionately less run_time at high sleep_avg to
@@ -3361,7 +3374,7 @@ need_resched_nonpreemptible:
+       delta = 0;
+
+       if (next->sleep_type == SLEEP_INTERACTIVE)
-           delta = delta * (ON_RUNQUEUE_WEIGHT * 128 / 100) / 128;
+           delta = delta * (on_runqueue_weight * 128 / 100) / 128;
+
+       array = next->array;
+       new_prio = recalc_task_prio(next, next->timestamp + delta);
@@ -6731,6 +6744,19 @@ void __init sched_init(void)
+ {
+     int i, j, k;
+
+     min_timeslice = DEFAULT_MIN_TIMESLICE;
+     def_timeslice = DEFAULT_DEF_TIMESLICE;
+     on_runqueue_weight = DEFAULT_ON_RUNQUEUE_WEIGHT;
+     child_penalty = DEFAULT_CHILD_PENALTY;
+     parent_penalty = DEFAULT_PARENT_PENALTY;
+     exit_weight = DEFAULT_EXIT_WEIGHT;
+     prio_bonus_ratio = DEFAULT_PRIO_BONUS_RATIO;
+     max_bonus = DEFAULT_MAX_BONUS;
+     interactive_delta = DEFAULT_INTERACTIVE_DELTA;
+     max_sleep_avg = DEFAULT_MAX_SLEEP_AVG;
+     starvation_limit = DEFAULT_STARVATION_LIMIT;
+     ns_max_sleep_avg = DEFAULT_NS_MAX_SLEEP_AVG;
+
+     for_each_possible_cpu(i) {
+         struct prio_array *array;
+         struct rq *rq;

```



```

+GENETIC_SCHED_TUNABLE(interactive_delta);
+
+struct gene_obj_attribute cpu_gene_obj_attrs[CPU_GENERAL_NUM_GENES] = {
+    GENETIC_TUNABLE_ATTR(child_penalty),
+    GENETIC_TUNABLE_ATTR(parent_penalty),
+    GENETIC_TUNABLE_ATTR(min_timeslice),
+    GENETIC_TUNABLE_ATTR(def_timeslice),
+    GENETIC_TUNABLE_ATTR(on_runqueue_weight),
+    GENETIC_TUNABLE_ATTR(exit_weight),
+    GENETIC_TUNABLE_ATTR(prio_bonus_ratio),
+    GENETIC_TUNABLE_ATTR(max_bonus),
+    GENETIC_TUNABLE_ATTR(starvation_limit),
+    GENETIC_TUNABLE_ATTR(max_sleep_avg),
+    GENETIC_TUNABLE_ATTR(ns_max_sleep_avg),
+    GENETIC_TUNABLE_ATTR(interactive_delta),
+};
+
+gene_param_t cpu_gene_param[CPU_GENERAL_NUM_GENES] = {
+    { "child_penalty",    DEFAULT_CHILD_PENALTY/3,
+      DEFAULT_CHILD_PENALTY*3, DEFAULT_CHILD_PENALTY, 0 },
+    { "parent_penalty",  DEFAULT_PARENT_PENALTY/3,
+      DEFAULT_PARENT_PENALTY*3, DEFAULT_PARENT_PENALTY, 0 },
+    { "min_timeslice",   1, 20, 10, 0 },
+    { "def_timeslice",   DEFAULT_DEF_TIMESLICE/3,
+      DEFAULT_DEF_TIMESLICE*3, DEFAULT_DEF_TIMESLICE, 0 },
+    { "on_runqueue_weight", DEFAULT_ON_RUNQUEUE_WEIGHT/3,
+      DEFAULT_ON_RUNQUEUE_WEIGHT*3, DEFAULT_ON_RUNQUEUE_WEIGHT, 0 },
+    { "exit_weight",     DEFAULT_EXIT_WEIGHT/3,
+      DEFAULT_EXIT_WEIGHT*3, DEFAULT_EXIT_WEIGHT, 0 },
+    { "prio_bonus_ratio", DEFAULT_PRIO_BONUS_RATIO/3,
+      DEFAULT_PRIO_BONUS_RATIO*3, DEFAULT_PRIO_BONUS_RATIO, 0 },
+    { "max_bonus",       DEFAULT_MAX_BONUS/3,
+      DEFAULT_MAX_BONUS*3, DEFAULT_MAX_BONUS, 0 },
+    { "starvation_limit", DEFAULT_STARVATION_LIMIT/3,
+      DEFAULT_STARVATION_LIMIT*3, DEFAULT_STARVATION_LIMIT, 0 },
+    { "max_sleep_avg",   DEFAULT_MAX_SLEEP_AVG/3,
+      DEFAULT_MAX_SLEEP_AVG*3, DEFAULT_MAX_SLEEP_AVG, 0 },
+    { "ns_max_sleep_avg", DEFAULT_NS_MAX_SLEEP_AVG/2,
+      (DEFAULT_NS_MAX_SLEEP_AVG/2) + DEFAULT_NS_MAX_SLEEP_AVG,
+      DEFAULT_NS_MAX_SLEEP_AVG, 0 },
+    { "interactive_delta", 1,
+      DEFAULT_INTERACTIVE_DELTA*3, DEFAULT_INTERACTIVE_DELTA, 0 },
+};
+
+static void cpu_take_stats_snapshot(phenotype_t * pt);
+static void cpu_general_create_child(genetic_child_t * child);
+static void cpu_general_set_child_genes(void * in_genes);
+static void cpu_throughput_create_child(genetic_child_t * child);
+static void cpu_latency_create_child(genetic_child_t * child);
+static void cpu_context_switch_create_child(genetic_child_t * child);
+static void cpu_throughput_calc_fitness(genetic_child_t * child);
+static void cpu_latency_calc_fitness(genetic_child_t * child);
+static void cpu_context_switch_calc_fitness(genetic_child_t * child);
+static void cpu_general_calc_post_fitness(phenotype_t * in_pt);
+static void cpu_shift_mutation_rate(phenotype_t * in_pt);
+
+struct sched_info * cpu_stats_snapshot;
+
+struct genetic_ops cpu_general_genetic_ops = {
+    .create_child = cpu_general_create_child,
+    .set_child_genes = cpu_general_set_child_genes,
+    .combine_genes = genetic_generic_combine_genes,
+    .mutate_child = genetic_generic_mutate_child,
+    .calc_post_fitness = cpu_general_calc_post_fitness,
+    .take_snapshot = cpu_take_stats_snapshot,
+    .shift_mutation_rate = cpu_shift_mutation_rate,
+};
+
+struct genetic_ops cpu_throughput_genetic_ops = {
+    .create_child = cpu_throughput_create_child,
+    .calc_fitness = cpu_throughput_calc_fitness,
+};
+
+struct genetic_ops cpu_latency_genetic_ops = {
+    .create_child = cpu_latency_create_child,
+    .calc_fitness = cpu_latency_calc_fitness,
+};
+
+struct genetic_ops cpu_context_switch_genetic_ops = {
+    .create_child = cpu_context_switch_create_child,
+    .calc_fitness = cpu_context_switch_calc_fitness,
+};
+
+#endif
+
+/*
+ * If a task is 'interactive' then we reinsert it in the active
+@@ -302,6 +426,8 @@ static DEFINE_PER_CPU(struct rq, runqueue
+ #define finish_arch_switch(prev) do { } while (0)
+ #endif
+
+
+
+#ifndef __ARCH_WANT_UNLOCKED_CTXSW
+static inline int task_running(struct rq *rq, struct task_struct *p)
+{
+@@ -6910,3 +7036,287 @@ void set_curr_task(int cpu, struct task_
+ }
+
+#endif

```

```

+
+#ifdef CONFIG_GENETIC_CPU_SCHED
+#define CPU_REGISTER_GENE(_gp, _name, _min, _max, _init, _mutate)\
+sysfs_create_file(&_gp.kobj, &sched_attr_##_name.attr);
+
+#define CHECK_PHENOTYPE(_pt)
+if(_pt == NULL) panic("%s: failed to register phenotype", \
+    __FUNCTION__);
+
+static int genetic_cpu_sched_init(void)
+{
+    int ret, i = 0;
+    genetic_t * genetic = NULL;
+    phenotype_t * pt;
+
+    cpu_stats_snapshot = (struct sched_info *)kmalloc(sizeof(struct sched_info), GFP_KERNEL);
+
+    if (!cpu_stats_snapshot)
+        panic("%s: failed to malloc enough space", __FUNCTION__);
+
+    ret = genetic_init(&genetic,
+                      CPU_NUM_CHILDREN,
+                      8 * HZ,
+                      0,
+                      "ohone-cpuscheduler");
+
+    if (ret)
+        panic("%s: failed to init genetic lib",
+              __FUNCTION__);
+
+    pt = genetic_register_phenotype(genetic,
+                                    &cpu_throughput_genetic_ops,
+                                    CPU_NUM_CHILDREN,
+                                    "throughput",
+                                    CPU_THROUGHPUT_NUM_GENES,
+                                    CPU_THROUGHPUT_UID);
+
+    CHECK_PHENOTYPE(pt);
+
+    pt = genetic_register_phenotype(genetic,
+                                    &cpu_latency_genetic_ops,
+                                    CPU_NUM_CHILDREN,
+                                    "latency",
+                                    CPU_LATENCY_NUM_GENES,
+                                    CPU_LATENCY_UID);
+
+    CHECK_PHENOTYPE(pt);
+
+    pt = genetic_register_phenotype(genetic,
+                                    &cpu_context_switch_genetic_ops,
+                                    CPU_NUM_CHILDREN,
+                                    "context-switch",
+                                    CPU_CONTEXT_SWITCH_NUM_GENES,
+                                    CPU_CONTEXT_SWITCH_UID);
+
+    CHECK_PHENOTYPE(pt);
+
+    pt = genetic_register_phenotype(genetic,
+                                    &cpu_general_genetic_ops,
+                                    CPU_NUM_CHILDREN,
+                                    "general",
+                                    CPU_GENERAL_NUM_GENES,
+                                    CPU_GENERAL_UID);
+
+    CHECK_PHENOTYPE(pt);
+
+    for (i = 0; i < CPU_GENERAL_NUM_GENES; i++) {
+        genetic_gene_obj_create(&cpu_gene_param[i], genetic,
+                                &cpu_gene_obj_attrs[i], &pt->genes);
+    }
+
+    genetic_start(genetic);
+
+    return 0;
+}
+postcore_initcall(genetic_cpu_sched_init);
+
+#define child_stats(child) ((struct sched_info *) (child)->stats_snapshot)
+
+static void cpu_take_stats_snapshot(phenotype_t * pt)
+{
+    int cpu;
+    struct sched_info * ss = child_stats(pt->child_ranking[0]);
+
+    memset(ss, 0, sizeof(struct sched_info));
+
+    for_each_online_cpu(cpu) {
+        struct rq *rq = cpu_rq(cpu);
+
+        ss->cpu_time += rq->rq_sched_info.cpu_time;
+        ss->run_delay += rq->rq_sched_info.run_delay;
+        ss->pcnt += rq->rq_sched_info.pcnt;
+    }
+}
+
+static void cpu_general_create_child(genetic_child_t * child)
+{
+    BUG_ON(!child);
+}

```

```

+ child->genes = (void *)kmalloc(sizeof(struct cpu_genes), GFP_KERNEL);
+ if (!child->genes)
+     panic("cpu_general_create_child: error mallocing space");
+
+ child->num_genes = CPU_GENERAL_NUM_GENES;
+ child->gene_param = cpu_gene_param;
+ child->stats_snapshot = cpu_stats_snapshot;
+
+ genetic_create_child_defaults(child);
+}
+
+static void cpu_general_set_child_genes(void * in_genes)
+{
+     struct cpu_genes * genes = (struct cpu_genes *)in_genes;
+
+     on_runqueue_weight = genes->on_runqueue_weight;
+     exit_weight = genes->exit_weight;
+     prio_bonus_ratio = genes->prio_bonus_ratio;
+     max_bonus = genes->max_bonus;
+     min_timeslice = genes->min_timeslice;
+
+     if (genes->def_timeslice < min_timeslice)
+         def_timeslice = genes->min_timeslice;
+     else
+         def_timeslice = genes->def_timeslice;
+
+     max_sleep_avg = def_timeslice * max_bonus;
+     interactive_delta = genes->interactive_delta;
+     child_penalty = genes->child_penalty;
+     parent_penalty = genes->parent_penalty;
+     ns_max_sleep_avg = JIFFIES_TO_NS(max_sleep_avg);
+     starvation_limit = max_sleep_avg;
+}
+
+static void inline cpu_create_child(genetic_child_t * child)
+{
+     BUG_ON(!child);
+
+     child->genes = 0;
+     child->gene_param = 0;
+     child->num_genes = 0;
+     child->stats_snapshot = cpu_stats_snapshot;
+}
+
+static void cpu_throughput_create_child(genetic_child_t * child)
+{
+     cpu_create_child(child);
+}
+
+static void cpu_latency_create_child(genetic_child_t * child)
+{
+     cpu_create_child(child);
+}
+
+static void cpu_context_switch_create_child(genetic_child_t * child)
+{
+     cpu_create_child(child);
+}
+
+static void cpu_throughput_calc_fitness(genetic_child_t * child)
+{
+     int cpu;
+     unsigned int cpu_time = 0;
+     struct sched_info * ss = child_stats(child);
+
+     for_each_online_cpu(cpu) {
+         struct rq *rq = cpu_rq(cpu);
+
+         cpu_time += rq->rq_sched_info.cpu_time;
+     }
+
+     child->fitness = cpu_time - ss->cpu_time;
+}
+
+static void cpu_latency_calc_fitness(genetic_child_t * child)
+{
+     int cpu;
+     unsigned int run_delay = 0;
+     struct sched_info * ss = child_stats(child);
+
+     for_each_online_cpu(cpu) {
+         struct rq *rq = cpu_rq(cpu);
+
+         run_delay += rq->rq_sched_info.run_delay;
+     }
+
+     child->fitness = (run_delay - ss->run_delay);
+     child->fitness = -child->fitness;
+}
+
+static void cpu_context_switch_calc_fitness(genetic_child_t * child)
+{
+     int cpu;
+     unsigned int pont = 0;
+     struct sched_info * ss = child_stats(child);
+
+     for_each_online_cpu(cpu) {
+         struct rq *rq = cpu_rq(cpu);
+

```

```

+         pcnt += rq->rq_sched_info.pcnt;
+     }
+
+     child->fitness = (pcnt - ss->pcnt);
+     child->fitness = -child->fitness;
+ }
+
+static void cpu_general_calc_post_fitness(phenotype_t * in_pt)
+{
+     struct list_head * entry;
+     phenotype_t * pt;
+     genetic_t * genetic = to_phenotype_genetic(in_pt);
+     int ranking[CPU_NUM_CHILDREN];
+     int weight = 1;
+     int i;
+
+     memset(ranking, 0, sizeof(ranking));
+
+     list_for_each(entry, &genetic->phenotypes.list) {
+         pt = to_phenotype(to_kobj(entry));
+
+         /* Look at everyone else that contributes to this
+          * phenotype */
+         if (pt->uid & CPU_GENERAL_UID && pt->uid != CPU_GENERAL_UID) {
+
+             switch (pt->uid) {
+             case CPU_CONTEXT_SWITCH_UID:
+                 weight = 2;
+                 break;
+             case CPU_THROUGHPUT_UID:
+                 weight = 2;
+                 break;
+             case CPU_LATENCY_UID:
+                 weight = 1;
+                 break;
+             default:
+                 BUG();
+             }
+
+             for (i = 0; i < pt->num_children; i++)
+                 ranking[pt->child_ranking[i]->id] += (i * weight);
+         }
+     }
+
+     for (i = 0; i < in_pt->num_children; i++)
+         in_pt->child_ranking[i]->fitness = ranking[i];
+ }
+
+static void cpu_shift_mutation_rate(phenotype_t * in_pt)
+{
+     struct list_head * entry;
+     struct genetic_s * g = to_phenotype_genetic(in_pt);
+     phenotype_t * pt;
+     int count = 0;
+     long rate = 0;
+
+     list_for_each(entry, &g->phenotypes.list) {
+         pt = to_phenotype(to_kobj(entry));
+
+         /* Look at everyone else that contributes to this
+          * phenotype */
+         if (pt->uid & CPU_GENERAL_UID && pt->uid != CPU_GENERAL_UID) {
+
+             switch (pt->uid) {
+             case CPU_CONTEXT_SWITCH_UID:
+             case CPU_THROUGHPUT_UID:
+             case CPU_LATENCY_UID:
+                 rate += pt->mutation_rate;
+                 count++;
+                 break;
+             default:
+                 BUG();
+             }
+         }
+     }
+
+     /* If we are a general phenotype that is made up of other
+      * phenotypes then we take the average */
+     if (count)
+         in_pt->mutation_rate = (rate / count);
+     else
+         BUG();
+ }
+
+##endif
+Index: linux-gl/init/Kconfig
+-----
+--- linux-gl.orig/init/Kconfig
++++ linux-gl/init/Kconfig
+@@ -182,6 +182,14 @@ config TASK_DELAY_ACCT
+
+     Say N if unsure.
+
+config GENETIC_CPU_SCHED
+bool "Genetic CPU scheduler (EXPERIMENTAL)"
+default y
+depends on GENETIC_LIB && SCHEDSTATS && EXPERIMENTAL
+help
+This will tune the O(1) CPU Scheduler tunables dynamically based on
+workload statistics.

```

Sep 05, 06 0:16

genetic-cpu-sched.patch

Page 6/6

```
config SYSCTL
    bool "Sysctl support" if EMBEDDED
    default y
@@ -509,6 +517,7 @@ config STOP_MACHINE
    depends on (SMP && MODULE_UNLOAD) || HOTPLUG_CPU
    help
        Need stop_machine() primitive.
+
endmenu

menu "Block layer"
Index: linux-gl/lib/Kconfig.debug
=====
--- linux-gl.orig/lib/Kconfig.debug
+++ linux-gl/lib/Kconfig.debug
@@ -84,6 +84,7 @@ config DETECT_SOFTLOCKUP
config SCHEDSTATS
    bool "Collect scheduler statistics"
    depends on DEBUG_KERNEL && PROC_FS
+
    default y
    help
        If you say Y here, additional code will be inserted into the
        scheduler and related routines to collect statistics about
```

```

block: General purpose routines to fingerprint disk IO

Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>
Index: linux-rc/block/genhd.c
=====
--- linux-rc.orig/block/genhd.c 2006-07-31 12:18:46.000000000 -0400
+++ linux-rc/block/genhd.c      2006-07-31 12:18:54.000000000 -0400
@@ -29,6 +29,8 @@
     char name[16];
 } *major_names[BLKDEV_MAJOR_HASH_SIZE];

+LIST_HEAD(gendisks);
+
/* index in the above - for now: assume no multimajor ranges */
static inline int major_to_index(int major)
{
@@ -387,19 +389,22 @@
     jiffies_to_msecs(disk_stat_read(disk, io_ticks)),
     jiffies_to_msecs(disk_stat_read(disk, time_in_queue)));
}
+
+#ifdef CONFIG_FINGERPRINTING
static ssize_t disk_fp_read(struct gendisk * disk, char *page)
{
-    return sprintf(page, "reads: %llx\n"
-                       "writes: %llx\n"
-                       "head_pos: %llx\n"
-                       "avg_dist: %llx\n"
-                       "avg_size: %llx\n",
+    return sprintf(page, "reads: %lld\n"
+                       "writes: %lld\n"
+                       "head_pos: %lld\n"
+                       "avg_dist: %lld\n"
+                       "avg_size: %lld\n",
                    (unsigned long long)disk->fp_ss->reads,
                    (unsigned long long)disk->fp_ss->writes,
                    (unsigned long long)disk->fp_ss->head_pos,
                    (unsigned long long)disk->fp_ss->avg_dist,
                    (unsigned long long)disk->fp_ss->avg_size);
}
+#endif

static struct disk_attribute disk_attr_uevent = {
    .attr = { .name = "uevent", .mode = S_IWUSR },
@@ -425,10 +430,13 @@
    .attr = { .name = "stat", .mode = S_IRUGO },
    .show = disk_stats_read
};
+
+#ifdef CONFIG_FINGERPRINTING
static struct disk_attribute disk_attr_fp = {
    .attr = { .name = "fp", .mode = S_IRUGO },
    .show = disk_fp_read
};
+#endif

static struct attribute * default_attrs[] = {
    &disk_attr_uevent.attr,
@@ -437,7 +445,9 @@
    &disk_attr_removable.attr,
    &disk_attr_size.attr,
    &disk_attr_stat.attr,
+#ifdef CONFIG_FINGERPRINTING
    &disk_attr_fp.attr,
+#endif
    NULL,
};

@@ -446,6 +456,7 @@
struct gendisk *disk = to_disk(kobj);
kfree(disk->random);
kfree(disk->part);
+ list_del(&disk->gendisks);
free_disk_stats(disk);
kfree(disk);
}

@@ -646,6 +657,7 @@
    kobj_set_kset_s(disk, block_subsys);
    kobject_init(&disk->kobj);
    rand_initialize_disk(disk);
+ list_add_tail(&disk->gendisks, &gendisks);
}

    disk->fp_ss = kmalloc(sizeof(struct fp_snapshot), GFP_KERNEL);
Index: linux-rc/block/ll_rw_blk.c
=====
--- linux-rc.orig/block/ll_rw_blk.c 2006-07-31 12:18:46.000000000 -0400
+++ linux-rc/block/ll_rw_blk.c 2006-07-31 12:18:54.000000000 -0400
@@ -21,6 +21,7 @@
#include <linux/string.h>
#include <linux/init.h>
#include <linux/bootmem.h> /* for max_pfn/max_low_pfn */
+#include <linux/genetic.h>
#include <linux/completion.h>
#include <linux/slab.h>
#include <linux/swap.h>
@@ -2608,6 +2609,141 @@
    __elv_add_request(q, req, ELEVATOR_INSERT_SORT, 0);
}

```

```

#ifdef CONFIG_GENETIC_IOSCHED_AS
extern struct list_head gendisks;
+
+void disk_stats_snapshot(phenotype_t * pt)
+{
+    struct list_head * d;
+    struct gendisk *disk;
+    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)pt->child_ranking[0]->stats_snapshot;
+
+    memset(ss, 0, sizeof(struct disk_stats_snapshot));
+
+    list_for_each(d, &gendisks) {
+        disk = list_entry(d, struct gendisk, gendisks);
+
+        disk_round_stats(disk);
+
+        ss->reads += disk_stat_read(disk, ios[READ]);
+        ss->writes += disk_stat_read(disk, ios[WRITE]);
+        ss->read_sectors += disk_stat_read(disk, sectors[READ]);
+        ss->write_sectors += disk_stat_read(disk, sectors[WRITE]);
+        ss->time_in_queue += disk_stat_read(disk, time_in_queue);
+    }
+}
+
+long long disk_num_ops_calc_fitness(genetic_child_t * child)
+{
+    struct list_head * d;
+    struct gendisk *disk;
+    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)child->stats_snapshot;
+    long long reads = 0;
+    long long writes = 0;
+
+    list_for_each(d, &gendisks) {
+        disk = list_entry(d, struct gendisk, gendisks);
+
+        disk_round_stats(disk);
+
+        reads += disk_stat_read(disk, ios[READ]);
+        writes += disk_stat_read(disk, ios[WRITE]);
+    }
+
+    reads -= ss->reads;
+    writes -= ss->writes;
+
+    return reads + writes;
+}
+
+long long disk_throughput_calc_fitness(genetic_child_t * child)
+{
+    struct list_head * d;
+    struct gendisk *disk;
+    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)child->stats_snapshot;
+    long long read_sectors = 0;
+    long long write_sectors = 0;
+
+    list_for_each(d, &gendisks) {
+        disk = list_entry(d, struct gendisk, gendisks);
+
+        disk_round_stats(disk);
+
+        read_sectors += disk_stat_read(disk, sectors[READ]);
+        write_sectors += disk_stat_read(disk, sectors[WRITE]);
+    }
+
+    read_sectors -= ss->read_sectors;
+    write_sectors -= ss->write_sectors;
+
+    return read_sectors + write_sectors;
+}
+
+long long disk_latency_calc_fitness(genetic_child_t * child)
+{
+    struct list_head * d;
+    struct gendisk *disk;
+    struct disk_stats_snapshot * ss = (struct disk_stats_snapshot *)child->stats_snapshot;
+    long long time_in_queue = 0;
+
+    list_for_each(d, &gendisks) {
+        disk = list_entry(d, struct gendisk, gendisks);
+
+        disk_round_stats(disk);
+
+        time_in_queue += disk_stat_read(disk, time_in_queue);
+    }
+
+    time_in_queue = -(time_in_queue - ss->time_in_queue);
+
+    return time_in_queue;
+}
+
#ifdef CONFIG_FINGERPRINTING
+
+void disk_update_fingerprint(phenotype_t * pt)
+{
+    struct list_head * d;
+    struct gendisk *disk;
+
+    BUG_ON(!pt->fp_ss);
+
+    /* tally up all the other disk snapshots */

```

```

+   list_for_each(d, &gendisks) {
+       disk = list_entry(d, struct gendisk, gendisks);
+
+       consolidate_fp_snapshot(pt->fp_ss, disk->fp_ss);
+
+       /* reset it for the next generation */
+       reset_fp_snapshot(disk->fp_ss);
+   }
+}
+
+void disk_get_fingerprint(phenotype_t * pt)
+{
+   struct list_head * d;
+   struct gendisk *disk;
+
+   BUG_ON(!pt->fp_ss);
+
+   /* tally up all the other disk snapshots */
+   list_for_each(d, &gendisks) {
+       disk = list_entry(d, struct gendisk, gendisks);
+
+       consolidate_fp_snapshot(pt->fp_ss, disk->fp_ss);
+
+       /* reset it for the next generation */
+       reset_fp_snapshot(disk->fp_ss);
+   }
+
+   calc_fp(pt->fp, pt->fp_ss);
+}
+
+##endif /* CONFIG_FINGERPRINTING */
+
+##endif /* GENETIC_IOSCHED_AS */
+
+/*
+ * disk_round_stats() - Round off the performance stats on a struct
+ * disk_stats.
+Index: linux-rc/include/linux/genhd.h
+=====
+--- linux-rc.orig/include/linux/genhd.h 2006-07-31 12:18:46.000000000 -0400
+++ linux-rc/include/linux/genhd.h 2006-07-31 12:18:54.000000000 -0400
+@@ -124,6 +124,7 @@
+     atomic_t sync_io;          /* RAID */
+     unsigned long stamp;
+     int in_flight;
+ + struct list_head gendisks;
+ #ifdef CONFIG_SMP
+     struct disk_stats *dkstats;
+ #else
+Index: linux-rc/include/linux/blkdev.h
+=====
+--- linux-rc.orig/include/linux/blkdev.h 2006-07-31 12:14:46.000000000 -0400
+++ linux-rc/include/linux/blkdev.h 2006-07-31 12:18:54.000000000 -0400
+@@ -833,12 +833,23 @@
+     _res; \
+ } \
+ )
+##endif
+##endif
+
+#define MODULE_ALIAS_BLOCKDEV(major,minor) \
+MODULE_ALIAS("block-major-" __stringify(major) "-" __stringify(minor))
+#define MODULE_ALIAS_BLOCKDEV_MAJOR(major) \
+MODULE_ALIAS("block-major-" __stringify(major) "-*")
+
+##ifdef CONFIG_GENETIC_IOSCHED_AS
+ +
+ +struct disk_stats_snapshot
+ +{
+ +   unsigned long reads;
+ +   unsigned long writes;
+ +   unsigned long read_sectors;
+ +   unsigned long write_sectors;
+ +   unsigned long time_in_queue;
+ +};
+##endif /* CONFIG_GENETIC_IOSCHED_AS */
+
+##endif

```

Sep 05, 06 0:16

genetic-lib.patch

Page 1/21

genetic-lib: Genetic Library for Runtime Kernel Tuning

The primary goal of the genetic library is to provide a framework for tuning various constants in the Kernel by evaluating the current workload and performance of the system. More information about the performance improvements and implementation of the library can be found in the 2005/2006 OLS papers:

http://www.linuxsymposium.org/2005/linuxsymposium_procv1.pdf pg 335

http://www.linuxsymposium.org/2006/linuxsymposium_procv2.pdf pg 173

Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>

Signed-off-by: Brandon Philips <philips@vnet.ibm.com>

```
---
include/linux/fingerprinting.h | 125 ++++
include/linux/genetic.h       | 351 ++++++
lib/Kconfig                   | 7
lib/Makefile                   | 1
lib/fingerprinting.c          | 276 ++++++
lib/genetic.c                  | 1183 ++++++
6 files changed, 1943 insertions(+)
```

Index: linux-gl/lib/Kconfig

```
-----
--- linux-gl.orig/lib/Kconfig
+++ linux-gl/lib/Kconfig
@@ -38,6 +38,13 @@ config LIBCRC32C
     require M here. See Castagnoli93.
     Module will be libcrc32c.
```

```
+config GENETIC_LIB
+   bool "Genetic Library"
+   default y
+   help
+       This option will build in a genetic library that will tweak
+       kernel parameters autonomically to improve performance.
```

```
#
# compression support is select'ed if needed
#
```

Index: linux-gl/lib/Makefile

```
-----
--- linux-gl.orig/lib/Makefile
+++ linux-gl/lib/Makefile
@@ -37,6 +37,7 @@ obj-$(CONFIG_CRC_CCITT) += crc-ccitt.o
obj-$(CONFIG_CRC16) += crc16.o
obj-$(CONFIG_CRC32) += crc32.o
obj-$(CONFIG_LIBCRC32C) += libcrc32c.o
+obj-$(CONFIG_GENETIC_LIB) += genetic.o
obj-$(CONFIG_GENERIC_IOMAP) += iomap.o
obj-$(CONFIG_GENERIC_ALLOCATOR) += genalloc.o
```

Index: linux-gl/lib/genetic.c

```
-----
--- /dev/null
+++ linux-gl/lib/genetic.c
@@ -0,0 +1,1183 @@
+/*
+ * Genetic Algorithm Library
+ *
+ * Jake Moilanen <moilanen@austin.ibm.com>
+ * Brandon Philips <brandon@ifup.org>
+ * Copyright (C) 2004-2006 IBM
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2 of the GNU General Public License as published
+ * by the Free Software Foundation.
+ */
+
+/*
+ * Life cycle
+ *
+ * 1.) Create random children
+ * 2.) Run tests
+ * 3.) Calculate fitness
+ * 4.) Take top performers
+ * 5.) Make children
+ * 6.) Mutate
+ * 7.) Goto step 2
+ */
+
+/*
+ * TODO:
+ *
+ * - Check to make sure DEF_DESKTOP_TIMESLICE is operating correctly
+ * - fix fixup_timeslice
+ */
+
+#include <linux/genetic.h>
+#include <linux/timer.h>
+#include <linux/jiffies.h>
+#include <linux/debugfs.h>
+#include <linux/init.h>
+#include <linux/random.h>
+#include <asm/uaccess.h>
+#include <asm/string.h>
+#include <asm/bug.h>
+
+#ifdef CONFIG_FINGERPRINTING
+#include <linux/fingerprinting.h>
+#include "fingerprinting.c"
```

Tuesday September 05, 2006

43/68

```

#endif
+
#define GENETIC_DEBUG_VERBOSE 1
+
+char genetic_lib_version[] = "0.3.1";
+
+int mutation_rate_change = GENETIC_DEFAULT_MUTATION_RATE_CHANGE;
+int genetic_lib_enabled = 1;
+
+decl_subsys(genetic, NULL, NULL);
+
+void dump_children(phenotype_t * pt);
+static void genetic_ns_top_parents(phenotype_t *);
+static void genetic_ns_ancestor_top_parents(phenotype_t *);
+static int genetic_create_children(phenotype_t *);
+static void genetic_split_performers(phenotype_t *);
+static void genetic_mutate(phenotype_t *);
+static void genetic_run_child(genetic_t * genetic);
+static void genetic_new_generation(genetic_t * genetic);
+
+void genetic_switch_child(unsigned long data);
+
+#define GENETIC_ATTR(_name, _mode, _show, _store) \
+struct genetic_attribute genetic_attr_##_name = { \
+    .attr = { .name = __stringify(_name), .mode = _mode }, \
+    .show = _show, \
+    .store = _store, \
+};
+
+#define GENETIC_SHOW(_name, format_string) \
+static int g_show_##_name(struct genetic_s *g, char *buf) \
+{ \
+    return sprintf(buf, format_string, g->_name); \
+}
+
+GENETIC_SHOW(child_number, "%lu\n");
+GENETIC_ATTR(child_number, S_IRUGO, g_show_child_number, NULL);
+
+GENETIC_SHOW(child_life_time, "%lu\n");
+GENETIC_ATTR(child_life_time, S_IRUGO, g_show_child_life_time, NULL);
+
+GENETIC_SHOW(num_children, "%lu\n");
+GENETIC_ATTR(num_children, S_IRUGO, g_show_num_children, NULL);
+
+GENETIC_SHOW(generation_number, "%lu\n");
+GENETIC_ATTR(generation_number, S_IRUGO, g_show_generation_number, NULL);
+
+GENETIC_SHOW(enabled, "%i\n");
+static ssize_t g_store_enabled (struct genetic_s *g, const char *buf,
+    size_t count)
+{
+    ssize_t consumed = -EINVAL;
+
+    if ((count > 0) && (*buf == '0' || *buf == '1')) {
+        g->enabled = *buf == '1' ? 1 : 0;
+        consumed = count;
+    }
+    return consumed;
+}
+GENETIC_ATTR(enabled, 0644, g_show_enabled, g_store_enabled);
+
+#ifdef CONFIG_FINGERPRINTING
+static int show_fingerprinting(struct genetic_s *g, char *buf)
+{
+    return sprintf(buf, "%d\n", g->fingerprinting);
+}
+GENETIC_ATTR(fingerprinting, S_IRUGO, show_fingerprinting, NULL);
#endif
+
+static struct attribute * genetic_attrs[] = {
+    &genetic_attr_child_number.attr,
+    &genetic_attr_child_life_time.attr,
+    &genetic_attr_num_children.attr,
+    &genetic_attr_generation_number.attr,
+    &genetic_attr_enabled.attr,
+#ifdef CONFIG_FINGERPRINTING
+    &genetic_attr_fingerprinting.attr,
+#endif
+    NULL
+};
+
+static ssize_t
+genetic_attr_store(struct kobject * kobj, struct attribute * attr,
+    const char * buf, size_t count)
+{
+    struct genetic_attribute * g_attr = to_genetic_attr(attr);
+    struct genetic_s * g = to_genetic(kobj);
+    ssize_t ret = -EIO;
+
+    if (g_attr->store)
+        ret = g_attr->store(g, buf, count);
+    return ret;
+}
+
+static ssize_t genetic_attr_show(struct kobject *kobj,
+    struct attribute *attr, char *page)
+{

```

```

+ struct genetic_s *g = (struct genetic_s *)to_genetic(kobj);
+ struct genetic_attribute *g_attr = to_genetic_attr(attr);
+ ssize_t ret = -EIO;
+
+ if (g_attr->show) {
+     ret = g_attr->show(g, page);
+ }
+
+ return ret;
+}
+
+static struct sysfs_ops genetic_sysfs_ops = {
+ .show = &genetic_attr_show,
+ .store = &genetic_attr_store,
+};
+
+/* TODO: free all phenotypes, etc */
+static void genetic_release(struct kobject *kobj)
+{
+ struct genetic_s *g = to_genetic(kobj);
+ kfree(g);
+}
+
+static struct kobj_type genetic_ktype = {
+ .release = &genetic_release,
+ .sysfs_ops = &genetic_sysfs_ops,
+ .default_attrs = genetic_attrs,
+};
+
+static ssize_t
+pt_attr_store(struct kobject *, struct attribute *, const char *, size_t);
+static ssize_t pt_attr_show(struct kobject *, struct attribute *, char *);
+
+static ssize_t
+pt_attr_store(struct kobject * kobj, struct attribute * attr,
+ const char * buf, size_t count)
+{
+ struct phenotype_s *p = (struct phenotype_s *)to_phenotype(kobj);
+ struct phenotype_attribute *p_attr = to_phenotype_attr(attr);
+ ssize_t ret = -EIO;
+
+ if (p_attr->store)
+     ret = p_attr->store(p, buf, count);
+
+ return ret;
+}
+
+static ssize_t pt_attr_show(struct kobject *kobj,
+ struct attribute *attr, char *page)
+{
+ struct phenotype_s *p = (struct phenotype_s *)to_phenotype(kobj);
+ struct phenotype_attribute *p_attr = to_phenotype_attr(attr);
+ ssize_t ret = -EIO;
+
+ if (p_attr->show) {
+     ret = p_attr->show(p, page);
+ }
+
+ return ret;
+}
+
+static struct sysfs_ops phenotype_sysfs_ops = {
+ .show = &pt_attr_show,
+ .store = &pt_attr_store,
+};
+
+/* TODO: free all phenotypes, etc */
+static void phenotype_release(struct kobject *kobj)
+{
+ struct genetic_s *g = to_genetic(kobj);
+ kfree(g);
+}
+
+#define PHENOTYPE_ATTR(_name, _mode, _show, _store) \
+struct phenotype_attribute pt_attr_##_name = { \
+ .attr = { .name = __stringify(_name), .mode = _mode }, \
+ .show = _show, \
+ .store = _store, \
+};
+
+#define PHENOTYPE_SHOW(_name, format_string) \
+static int p_show_##_name(struct phenotype_s *p, char *buf) \
+{ \
+     return sprintf(buf, format_string, p->_name); \
+}
+
+PHENOTYPE_SHOW(child_number, "%lu\n");
+PHENOTYPE_ATTR(child_number, S_IRUGO, p_show_child_number, NULL);
+
+PHENOTYPE_SHOW(num_children, "%lu\n");
+PHENOTYPE_ATTR(num_children, S_IRUGO, p_show_num_children, NULL);
+
+PHENOTYPE_SHOW(natural_selection_cutoff, "%lu\n");
+PHENOTYPE_ATTR(natural_selection_cutoff, S_IRUGO, \
+ p_show_natural_selection_cutoff, NULL);
+
+PHENOTYPE_SHOW(num_mutations, "%lu\n");
+PHENOTYPE_ATTR(num_mutations, S_IRUGO, p_show_num_mutations, NULL);
+
+PHENOTYPE_SHOW(num_genes, "%lu\n");
+PHENOTYPE_ATTR(num_genes, S_IRUGO, p_show_num_genes, NULL);

```

```

+
+PHENOTYPE_SHOW(uid, "%lu\n");
+PHENOTYPE_ATTR(uid, S_IRUGO, p_show_uid, NULL);
+
+PHENOTYPE_SHOW(avg_fitness, "%lli\n");
+PHENOTYPE_ATTR(avg_fitness, S_IRUGO, p_show_avg_fitness, NULL);
+
+PHENOTYPE_SHOW(last_gen_avg_fitness, "%lli\n");
+PHENOTYPE_ATTR(last_gen_avg_fitness, S_IRUGO, p_show_last_gen_avg_fitness, NULL);
+
+PHENOTYPE_SHOW(from_top, "%lli\n");
+PHENOTYPE_ATTR(from_top, S_IRUGO, p_show_from_top, NULL);
+
+static struct attribute * phenotype_attrs[] = {
+    &pt_attr_child_number.attr,
+    &pt_attr_num_children.attr,
+    &pt_attr_natural_selection_cutoff.attr,
+    &pt_attr_num_mutations.attr,
+    &pt_attr_num_genes.attr,
+    &pt_attr_uid.attr,
+    &pt_attr_avg_fitness.attr,
+    &pt_attr_last_gen_avg_fitness.attr,
+    &pt_attr_from_top.attr,
+    NULL,
+};
+
+static struct kobj_type phenotype_ktype = {
+    .release = &phenotype_release,
+    .sysfs_ops = &phenotype_sysfs_ops,
+    .default_attrs = phenotype_attrs,
+};
+
+int genetic_init(genetic_t ** in_genetic, unsigned long num_children,
+                unsigned long child_life_time, int fingerprinting,
+                char * name)
+{
+    genetic_t * genetic;
+
+    if (!genetic_lib_enabled)
+        return 0;
+
+    printk(KERN_INFO "Initializing Genetic Library - version %s\n",
+           genetic_lib_version);
+
+    genetic = (genetic_t *)kzalloc(sizeof(genetic_t), GFP_KERNEL);
+    if (!genetic) {
+        printk(KERN_ERR "genetic_init: not enough memory\n");
+        return -ENOMEM;
+    }
+
+    genetic->kobj.kset = &genetic_subsys.kset;
+    genetic->kobj.ktype = &genetic_ktype;
+    kobject_set_name(&genetic->kobj, name);
+
+    kobject_register(&genetic->kobj);
+    if (!(kobject_get(&genetic->kobj)))
+        return -ENOENT;
+
+    kobject_set_name(&genetic->phenotypes.kobj, "phenotypes");
+    genetic->phenotypes.ktype = &phenotype_ktype;
+    genetic->phenotypes.kobj.parent = &genetic->kobj;
+    kset_register(&genetic->phenotypes);
+
+    genetic->num_children = num_children;
+    genetic->child_life_time = child_life_time;
+
+    genetic->generation_number = 1;
+    genetic->child_number = 0;
+    genetic->enabled = 1;
+
+    #ifdef CONFIG_FINGERPRINTING
+    genetic->fingerprinting = fingerprinting;
+    #endif
+
+    /* Setup how long each child has to live */
+    init_timer(&genetic->timer);
+    genetic->timer.function = genetic_switch_child;
+    genetic->timer.data = (unsigned long)genetic;
+
+    *in_genetic = genetic;
+
+    return 0;
+}
+
+struct phenotype_s * genetic_register_phenotype(genetic_t * genetic,
+        struct genetic_ops * ops, unsigned long num_children,
+        char * name, unsigned long num_genes, unsigned long uid)
+{
+    phenotype_t * pt;
+    int rc;
+
+    if (!genetic_lib_enabled)
+        return 0;
+
+    printk(KERN_INFO "Initializing %s's phenotype %s\n", genetic->kobj.name,
+           name);
+
+    pt = (phenotype_t *)kzalloc(sizeof(phenotype_t), GFP_KERNEL);
+    if (!genetic) {
+        printk(KERN_ERR "genetic_register_phenotype: not enough\
+memory\n");
+    }

```

Sep 05, 06 0:16

genetic-lib.patch

Page 5/21

```

+         return NULL;
+     }
+
+     pt->child_ranking = (genetic_child_t **)kmalloc(num_children * sizeof(genetic_child_t *), GFP_KERNEL);
+     if (!pt->child_ranking) {
+         printk(KERN_ERR "genetic_register_phenotype: not enough\
+             memory\n");
+         kfree(pt);
+         return NULL;
+     }
+
+     pt->kobj.kset = kset_get(&genetic->phenotypes);
+     kobject_set_name(&pt->kobj, name);
+     kobject_register(&pt->kobj);
+
+     kobject_set_name(&pt->genes.kobj, "genes");
+     pt->genes.ktype = &gene_param_ktype;
+     pt->genes.kobj.parent = &pt->kobj;
+     kset_register(&pt->genes);
+
+     if (!(kobject_get(&pt->kobj)))
+         return NULL;
+
+     INIT_LIST_HEAD(&pt->children_queue[0]);
+     INIT_LIST_HEAD(&pt->children_queue[1]);
+
+     pt->run_queue = &pt->children_queue[0];
+     pt->finished_queue = &pt->children_queue[1];
+
+     pt->ops = ops;
+     pt->num_children = num_children;
+
+     pt->mutation_rate = GENETIC_DEFAULT_MUTATION_RATE;
+     pt->natural_selection = genetic_ns_top_parents;
+     pt->natural_selection_cutoff = num_children / 2;
+     pt->avg_fitness = 0;
+     pt->last_gen_avg_fitness = 0;
+     pt->child_number = 0;
+
+     pt->uid = uid;
+     pt->num_genes = num_genes;
+
+     pt->top_fitness = 0;
+
+ #ifdef CONFIG_FINGERPRINTING
+     if (genetic->fingerprinting) {
+         if ((rc = genetic_init_fingerprinting(pt)) < 0)
+             return NULL;
+     }
+ #endif
+
+     /* Create some children */
+     rc = genetic_create_children(pt);
+     if (rc < 0)
+         return NULL;
+
+     return pt;
+ }
+
+ #define GENE_PARAM_SHOW(_name, format_string) \
+ static int gp_show_##_name(struct gene_obj *g, char *buf) \
+ { \
+     return sprintf(buf, format_string, g->gene_param->_name); \
+ }
+
+ GENE_PARAM_SHOW(min, "%lu\n");
+ GENE_PARAM_ATTR(min, S_IRUGO, gp_show_min, NULL);
+
+ GENE_PARAM_SHOW(max, "%lu\n");
+ GENE_PARAM_ATTR(max, S_IRUGO, gp_show_max, NULL);
+
+ GENE_PARAM_SHOW(initial, "%lu\n");
+ GENE_PARAM_ATTR(initial, S_IRUGO, gp_show_initial, NULL);
+
+ static struct attribute * gene_param_attrs[] = {
+     &gene_param_attr_min.attr,
+     &gene_param_attr_max.attr,
+     &gene_param_attr_initial.attr,
+     NULL
+ };
+
+ ssize_t gene_obj_attr_store(struct kobject * kobj, struct attribute * attr,
+     const char * buf, size_t count)
+ {
+     struct gene_obj_attribute * g_attr = to_gene_param_attr(attr);
+     struct gene_obj * g = to_gene_obj(kobj);
+     ssize_t ret = -EIO;
+
+     if (g_attr->store)
+         ret = g_attr->store(g, buf, count);
+
+     return ret;
+ }
+
+ ssize_t gene_obj_attr_show(struct kobject * kobj,
+     struct attribute * attr, char * page)
+ {
+     struct gene_obj_attribute * g_attr = to_gene_param_attr(attr);
+     struct gene_obj * g = to_gene_obj(kobj);
+     ssize_t ret = -EIO;

```

```

+     if (g_attr->show)
+         ret = g_attr->show(g, page);
+
+     return ret;
+ }
+
+ struct sysfs_ops gene_param_sysfs_ops = {
+     .show = &gene_obj_attr_show,
+     .store = &gene_obj_attr_store,
+ };
+
+ /* TODO: free all phenotypes, etc */
+ void gene_param_release(struct kobject *kobj)
+ {
+     struct genetic_s *g = to_genetic(kobj);
+     kfree(g);
+ }
+
+ struct kobj_type gene_param_ktype = {
+     .release = &gene_param_release,
+     .sysfs_ops = &genetic_sysfs_ops,
+     .default_attrs = gene_param_attrs,
+ };
+
+ int genetic_gene_obj_create(struct gene_param_s *gp, genetic_t *g,
+     struct gene_obj_attribute *attr, struct kset *kset)
+ {
+     struct gene_obj *go = kzalloc(sizeof(struct gene_obj), GFP_KERNEL);
+
+     go->gene_param = gp;
+     go->genetic = g;
+     go->kobj.kset = kset_get(kset);
+     kobject_set_name(&go->kobj, gp->name);
+     kobject_register(&go->kobj);
+
+     return sysfs_create_file(&go->kobj, &attr->attr);
+ }
+
+ void genetic_start(genetic_t *genetic)
+ {
+     if (!genetic_lib_enabled)
+         return;
+
+     genetic_run_child(genetic);
+     printk(KERN_INFO "%ld children started in %s genetic library\n",
+         genetic->num_children, genetic->kobj.name);
+ }
+
+ /* create some children, it is up to the lib user to come up w/ a good
+  * distro of genes for it's children */
+ static int genetic_create_children(phenotype_t *pt)
+ {
+     unsigned long i;
+     genetic_child_t *child;
+
+     for (i = 0; i < pt->num_children; i++) {
+         pt->child_ranking[i] = (genetic_child_t *)kmallocc(
+             sizeof(genetic_child_t), GFP_KERNEL);
+
+         if (!pt->child_ranking[i]) {
+             printk(KERN_ERR "genetic_create_child: not enough\
+                 memory\n");
+
+             for (i = i - 1; i >= 0; i--)
+                 kfree(pt->child_ranking[i]);
+
+             return -ENOMEM;
+         }
+
+         child = pt->child_ranking[i];
+
+         child->id = i;
+
+         pt->ops->create_child(child);
+
+         list_add_tail(&child->list, pt->run_queue);
+     }
+
+     return 0;
+ }
+
+ /* See how well child did and run the next one */
+ void genetic_switch_child(unsigned long data)
+ {
+     genetic_t *genetic = (genetic_t *)data;
+     genetic_child_t *child;
+
+     struct list_head *p;
+     phenotype_t *pt;
+
+     int new_generation = 0;
+
+     list_for_each(p, &genetic->phenotypes.list) {
+         pt = to_phenotype(to_kobj(p));
+
+         child = list_entry(pt->run_queue->next, genetic_child_t, list);
+
+         list_del(&child->list);
+     }
+ }

```

```

+         list_add_tail(&child->list, pt->finished_queue);
+
+         if (pt->ops->calc_fitness)
+             pt->ops->calc_fitness(child);
+
+         pt->child_ranking[pt->child_number++] = child;
+
+         /* See if need more children */
+         if (list_empty(pt->run_queue))
+             new_generation = 1;
+
+     }
+
+     genetic->child_number++;
+
+     if (new_generation)
+         genetic_new_generation(genetic);
+
+     genetic_run_child(genetic);
+ }
+
+ /* Set the child's genes for run */
+ void genetic_run_child(genetic_t * genetic)
+ {
+     struct list_head * p;
+     phenotype_t * pt;
+
+     genetic_child_t * child;
+     void * genes;
+
+     list_for_each(p, &genetic->phenotypes.list) {
+         pt = to_phenotype(to_kobj(p));
+
+         child = list_entry(pt->run_queue->next, genetic_child_t, list);
+
+         genes = child->genes;
+
+         /* not enabled every child goes to defaults */
+         if (!genetic->enabled)
+             genetic_create_child_defaults(child);
+
+         /* set the child genes if we are enabled */
+         if (pt->ops->set_child_genes)
+             pt->ops->set_child_genes(genes);
+
+         if (pt->ops->take_snapshot)
+             pt->ops->take_snapshot(pt);
+
+     }
+
+     /* set a timer interrupt */
+     genetic->timer.expires = jiffies + genetic->child_life_time;
+     add_timer(&genetic->timer);
+ }
+
+ /* This natural selection routine will take the top
+  * natural_select_cutoff and use them to make children for the next
+  * generation and keep the top half performers
+  *
+  * This assumes natural_select_cutoff is exactly half of num_children
+  * and num_children is a multiple of 4.
+  */
+ static void genetic_ns_top_parents(phenotype_t * pt)
+ {
+     unsigned long i,j,k = 0;
+     unsigned long num_children = pt->num_children;
+     unsigned long cutoff = num_children - pt->natural_selection_cutoff;
+
+     for (i = cutoff, j = num_children - 1; i < j; i++, j--, k += 2) {
+         /* create child A */
+         pt->ops->combine_genes(pt->child_ranking[i],
+                               pt->child_ranking[j],
+                               pt->child_ranking[k]);
+
+         /* create child B */
+         pt->ops->combine_genes(pt->child_ranking[i],
+                               pt->child_ranking[j],
+                               pt->child_ranking[k+1]);
+     }
+ }
+
+ /* This natural selection routine just has top parents populating
+  * bottom performers. */
+ static void genetic_ns_ward_top_parents(phenotype_t * pt)
+ {
+     unsigned long i;
+     unsigned long num_children = pt->num_children;
+     unsigned long cutoff = num_children - pt->natural_selection_cutoff;
+
+     for (i = 0; i < cutoff; i += 2) {
+         pt->ops->combine_genes(pt->child_ranking[num_children - 1],
+                               pt->child_ranking[num_children - 2],
+                               pt->child_ranking[i]);
+
+         pt->ops->combine_genes(pt->child_ranking[num_children - 1],
+                               pt->child_ranking[num_children - 2],
+                               pt->child_ranking[i+1]);
+     }
+ }

```

```

+    }
+}
+
+static inline void genetic_swap(genetic_child_t ** a, genetic_child_t ** b)
+{
+    genetic_child_t * tmp = *a;
+
+    *a = *b;
+    *b = tmp;
+}
+
+/* bubble sort */
+/* XXX change this to quick sort */
+static void genetic_split_performers(phenotype_t * pt)
+{
+    int i, j;
+
+    for (i = pt->num_children; i > 1; i--)
+    for (j = 0; j < i - 1; j++)
+        if (pt->child_ranking[j]->fitness > pt->child_ranking[j+1]->fitness)
+            genetic_swap(&pt->child_ranking[j], &pt->child_ranking[j+1]);
+}
+
+static void genetic_mutate(phenotype_t * pt)
+{
+    long child_entry = -1;
+    int i;
+
+    if (!pt->num_genes)
+        return;
+
+    for (i = 0; i < pt->num_mutations; i++) {
+        get_random_bytes(&child_entry, sizeof(child_entry));
+        child_entry = child_entry % pt->num_children;
+
+        pt->ops->mutate_child(pt->child_ranking[child_entry]);
+    }
+}
+
+/* XXX This will either aid in handling new workloads, or send us on a
+   downward spiral */
+static void genetic_shift_mutation_rate(phenotype_t * pt, long long prev_gen_avg_fitness, long long avg_fitness)
+{
+    long long low_bound;
+    long long high_bound;
+    long dummy;
+    struct genetic_s *g;
+
+    g = to_genetic(pt->kobj.parent);
+
+    if (mutation_rate_change && g->generation_number > 1) {
+
+        if (pt->ops->shift_mutation_rate) {
+            pt->ops->shift_mutation_rate(pt);
+        } else {
+
+            low_bound = avg_fitness * 90;
+            divll(&low_bound, 100, &dummy);
+
+            high_bound = avg_fitness * 110;
+            divll(&high_bound, 100, &dummy);
+
+            if (high_bound > prev_gen_avg_fitness)
+                pt->mutation_rate -= mutation_rate_change;
+            else if (low_bound < prev_gen_avg_fitness)
+                pt->mutation_rate += mutation_rate_change;
+
+            if (pt->mutation_rate > GENETIC_MAX_MUTATION_RATE)
+                pt->mutation_rate = GENETIC_MAX_MUTATION_RATE;
+            else if (pt->mutation_rate < GENETIC_MIN_MUTATION_RATE)
+                pt->mutation_rate = GENETIC_MIN_MUTATION_RATE;
+
+        }
+    }
+}
+
+void genetic_general_shift_mutation_rate(phenotype_t * in_pt)
+{
+    struct list_head * p;
+    phenotype_t * pt;
+    int count = 0;
+    long rate = 0;
+    struct kobject * k = &in_pt->kobj;
+
+    list_for_each(p, &k->kset->list) {
+        pt = to_phenotype(to_kobj(p));
+
+        if (in_pt->uid & pt->uid && in_pt->uid != pt->uid) {
+            rate += pt->mutation_rate;
+            count++;
+        }
+    }
+
+    /* If we are a general phenotype that is made up of other
+       phenotypes then we take the average */
+    if (count)
+        in_pt->mutation_rate = (rate / count);
+    else
+        in_pt->mutation_rate = mutation_rate_change;
+}

```

```

+
+
+static long long genetic_calc_stats_parent(phenotype_t * in_pt)
+{
+    struct list_head * p;
+    struct kobject * k = &in_pt->kobj;
+    phenotype_t * pt;
+    int i = 0;
+    long long total_fitness = 0;
+    long dummy;
+#ifdef CONFIG_FINGERPRINTING
+    int fp = to_phenotype_genetic(in_pt)->fingerprinting;
+    int numerical_fp;
+#endif
+
+#ifdef CONFIG_FINGERPRINTING
+    if (fp)
+        numerical_fp = create_fingerprint(in_pt->fp);
+
+    /* do we want this???? */
+    if ((fp && (in_pt->last_fingerprint == numerical_fp)) || !fp) {
+#else
+    if (1) {
+#endif
+        list_for_each(p, &k->kset->list) {
+            pt = to_phenotype(to_kobj(p));
+
+            /* for each child */
+            if (in_pt->uid & pt->uid && in_pt->uid != pt->uid) {
+                if (pt->avg_fitness) {
+                    /* measure how far percentage-wise that we are from the top */
+                    pt->from_top = (pt->last_gen_avg_fitness - pt->avg_fitness) * 100;
+                    divll(&pt->from_top, (pt->avg_fitness > 0) ? pt->avg_fitness : -pt->avg_fitness, &dummy);
+
+                    total_fitness += pt->from_top;
+                }
+            }
+            i++;
+        }
+    } else {
+        /* XXX horrible horrible hack...but
+         * testing viability */
+        total_fitness = 0;
+        i = 1;
+    }
+
+    BUG_ON(!i);
+
+    in_pt->last_gen_avg_fitness = total_fitness;
+    divll(&in_pt->last_gen_avg_fitness, i, &dummy);
+
+    return total_fitness;
+}
+
+static void genetic_calc_stats(phenotype_t * pt)
+{
+    struct genetic_s * g;
+    long long total_fitness = 0;
+    long long prev_gen_avg_fitness = pt->last_gen_avg_fitness;
+    long long tmp_fitness;
+    long dummy;
+    int i = 0;
+
+    /* On a general phenotype, need to look at other metrics since
+     * the fitness is normalized. It always average the same. It
+     * assumes that this phenotype is registered last.
+     */
+    if (pt->ops->calc_post_fitness) {
+        total_fitness = genetic_calc_stats_parent(pt);
+    } else {
+        /* calculate the avg fitness for this generation and avg fitness
+         * so far */
+        for (i = 0; i < pt->num_children; i++)
+            total_fitness += pt->child_ranking[i]->fitness;
+
+        tmp_fitness = total_fitness >> long_log2(pt->num_children);
+        pt->last_gen_avg_fitness = tmp_fitness;
+    }
+
+    /* Mutation rate calibration */
+    genetic_shift_mutation_rate(pt, prev_gen_avg_fitness,
+        pt->last_gen_avg_fitness);
+
+    pt->num_mutations = ((pt->num_children * pt->num_genes) * pt->mutation_rate) / 100;
+
+    /* calc new avg fitness */
+    tmp_fitness = pt->last_gen_avg_fitness - pt->avg_fitness;
+    g = to_phenotype_genetic(pt);
+    divll(&tmp_fitness, g->generation_number, &dummy);
+    pt->avg_fitness += tmp_fitness;
+
+    pt->fitness_history[pt->fitness_history_index++ & GENETIC_HISTORY_MASK]
+        = pt->last_gen_avg_fitness;
+}
+
+

```

```

+
+
+void genetic_new_generation(genetic_t * genetic)
+{
+    struct list_head * tmp;
+
+    struct list_head * p;
+    phenotype_t * pt;
+
+    list_for_each(p, &genetic->phenotypes.list) {
+        pt = to_phenotype(to_kobj(p));
+
+        /* Check to see if need to recalibrate fitness to take
+        other phenotypes' rankings into account. This
+        should be ran after all phenotypes that have input
+        have been ran. */
+        if (pt->ops->calc_post_fitness)
+            pt->ops->calc_post_fitness(pt);
+
+        dump_children(pt);
+
+        /* figure out top performers */
+        genetic_split_performers(pt);
+
+        /* calc stats */
+        genetic_calc_stats(pt);
+
+        dump_children(pt);
+
+        /* make some new children */
+        if (pt->num_genes)
+            pt->natural_selection(pt);
+
+        dump_children(pt);
+
+    #ifdef CONFIG_FINGERPRINTING
+        if (pt->ops->get_fingerprint) {
+
+            pt->ops->get_fingerprint(pt);
+            reset_fp_snapshot(pt->fp_ss);
+
+            /* See if this generation was a top performer
+            * for the current workload.
+            * Do this after natural selection to get rid
+            * of the bad apples
+            */
+            update_top_performers(pt);
+
+            /* We know the workload, lets put some known
+            good genes back in */
+            reintroduce_genes(pt);
+
+            pt->last_fingerprint = create_fingerprint(pt->fp);
+
+        }
+
+        dump_children(pt);
+    #endif
+
+    /* mutate a couple of the next generation */
+    genetic_mutate(pt);
+
+    dump_children(pt);
+
+    /* Move the new children still sitting in the finished queue to
+    the run queue */
+    tmp = pt->run_queue;
+    pt->run_queue = pt->finished_queue;
+    pt->finished_queue = tmp;
+
+    pt->child_number = 0;
+    #if GENETIC_DEBUG
+    pt->debug_index = 0;
+    #endif
+
+    }
+
+    genetic->child_number = 0;
+    genetic->generation_number++;
+}
+
+/**
+ * genetic_generic_random_mutate_gene - mutate child's gene to value in range
+ * @child: child whose gene we are mutating
+ * @gene_num: gene index from gene_param to mutate; gene must be unsigned long
+ *
+ * Mutate a gene picking a random value within the gene range that was
+ * specified in @child->gene_param.
+ */
+void genetic_generic_random_mutate_gene(genetic_child_t * child,
+    unsigned long gene_num)
+{
+    unsigned long *genes = (unsigned long *)child->genes;
+    unsigned long min = child->gene_param[gene_num].min;
+    unsigned long max = child->gene_param[gene_num].max;
+    unsigned long gene_value;
+    unsigned long range = max - min + 1;
+
+    /* create a mutation value */
+    get_random_bytes(&gene_value, sizeof(gene_value));
+

```

```

+     gene_value = gene_value % range;
+
+     genes[gene_num] = min + gene_value;
+ }
+ /**
+  * genetic_generic_iterative_mutate_gene
+  * @child: child whose gene we are mutating
+  * @gene_num: gene index from gene_param to mutate; gene must be unsigned long
+  */
+ void genetic_generic_iterative_mutate_gene(genetic_child_t * child,
+     unsigned long gene_num)
+ {
+     unsigned long *genes = (unsigned long *)child->genes;
+     unsigned long min = child->gene_param[gene_num].min;
+     unsigned long max = child->gene_param[gene_num].max;
+     long change;
+     unsigned long old_value = genes[gene_num];
+     unsigned long new_value;
+     unsigned long range = max - min + 1;
+
+     /* If under 5, random might work better */
+     if (range < 5)
+         return genetic_generic_random_mutate_gene(child, gene_num);
+
+     /* get the % of change */
+     get_random_bytes(&change, sizeof(change));
+
+     change = change % GENETIC_ITERATIVE_MUTATION_RANGE;
+
+     new_value = ((long)(change * range) / (long)100) + old_value;
+
+     if (new_value > max)
+         new_value = max;
+     else if (new_value < min)
+         new_value = min;
+
+     genes[gene_num] = new_value;
+ }
+ /**
+  * genetic_generic_mutate_child - mutate random gene in child
+  * @child: child whose gene we are mutating.
+  *
+  * Select a random gene and mutate it either using either the mutate_gene
+  * callback specified in '&struct gene_param' OR if that is NULL then use
+  * 'genetic_generic_random_mutate_gene()'
+  */
+ void genetic_generic_mutate_child(genetic_child_t * child)
+ {
+     long gene_num = -1;
+
+     /* pick a random gene */
+     get_random_bytes(&gene_num, sizeof(gene_num));
+
+     if (gene_num < 0)
+         gene_num = -gene_num;
+
+     gene_num = gene_num % child->num_genes;
+
+     if (child->gene_param[gene_num].mutate_gene)
+         child->gene_param[gene_num].mutate_gene(child, gene_num);
+     else
+         genetic_generic_random_mutate_gene(child, gene_num);
+ }
+ /**
+  * genetic_generic_mutate_child - set all genes to their initial value
+  */
+ void genetic_create_child_defaults(genetic_child_t * child)
+ {
+     int i;
+     unsigned long * genes = child->genes;
+
+     for (i = 0; i < child->num_genes; i++) {
+         genes[i] = child->gene_param[i].initial;
+     }
+ }
+ void genetic_create_child_spread(genetic_child_t * child,
+     unsigned long num_children)
+ {
+     int i;
+     unsigned long range;
+     int range_incr;
+     int child_num = child->id;
+     long num_genes = child->num_genes;
+     unsigned long * genes = child->genes;
+
+     for (i = 0; i < num_genes; i++) {
+         range = child->gene_param[i].max - child->gene_param[i].min + 1;
+         range_incr = range / num_children;
+         if (range_incr)
+             genes[i] = child->gene_param[i].min +
+                 (range_incr * child_num);
+         else
+             genes[i] = child->gene_param[i].min +
+                 (child_num / (num_children / range));
+     }
+ }

```

```

+}
+
+##if 0
+/* Randomly pick which parent to use for each gene to create a child */
+void genetic_generic_combine_genes(genetic_child_t * parent_a,
+    genetic_child_t * parent_b,
+    genetic_child_t * child)
+{
+    unsigned long * genes_a = (unsigned long *)parent_a->genes;
+    unsigned long * genes_b = (unsigned long *)parent_b->genes;
+    unsigned long * child_genes = (unsigned long *)child->genes;
+
+    /* Assume parent_a and parent_b have same num_genes */
+    unsigned long num_genes = parent_a->num_genes;
+    int parent_selector;
+    int i;
+
+    get_random_bytes(&parent_selector, sizeof(parent_selector));
+
+    if ((sizeof(parent_selector) * 8) < num_genes)
+        BUG();
+
+    for (i = 0; i < num_genes; i++) {
+        /* Look at each bit to determine which parent to use */
+        if (parent_selector & 1) {
+            child_genes[i] = genes_a[i];
+        } else {
+            child_genes[i] = genes_b[i];
+        }
+        parent_selector >>= 1;
+    }
+}
+##else
+
+/**
+ * genetic_generic_combine_genes - create child using comb. of parent's genes
+ * @parent_a: gene doner
+ * @parent_b: gene doner
+ * @child: genes will be modified using a combination of a and b
+ */
+void genetic_generic_combine_genes(genetic_child_t * parent_a,
+    genetic_child_t * parent_b,
+    genetic_child_t * child)
+{
+    unsigned long * genes_a = (unsigned long *)parent_a->genes;
+    unsigned long * genes_b = (unsigned long *)parent_b->genes;
+    unsigned long * child_genes = (unsigned long *)child->genes;
+
+    /* Assume parent_a and parent_b have same num_genes */
+    unsigned long num_genes = parent_a->num_genes;
+    int percentage;
+    int i;
+
+    for (i = 0; i < num_genes; i++) {
+        get_random_bytes(&percentage, sizeof(percentage));
+
+        /* Get percentage */
+        percentage = percentage % 100;
+
+        if (percentage < 0)
+            percentage = -percentage;
+
+        /* Give child x% of parent A's genes value, plus
+         * 100-x% of parent B's genes value */
+        child_genes[i] = (((genes_a[i]+1) * percentage) +
+            (genes_b[i] * (100 - percentage))) / 100;
+    }
+}
+##endif
+
+##if GENETIC_DEBUG
+/* Stores attributes into an array in the following format
+ * child_num fitness gene[0] gene[1] ... gene[num_genes-1]
+ * Add +1 to GENETIC_NUM_DEBUG_POINTS if add another dump_children
+ * call
+ */
+void dump_children(phenotype_t * pt)
+{
+    int i, j;
+    long * genes;
+    unsigned long debug_size = pt->debug_size;
+
+    for (i = 0; i < pt->num_children; i++) {
+        pt->debug_history[pt->debug_index++ % debug_size] = pt->child_ranking[i]->id;
+        pt->debug_history[pt->debug_index++ % debug_size] = pt->child_ranking[i]->fitness;
+
+        genes = (long *)pt->child_ranking[i]->genes;
+
+        for (j = 0; j < pt->child_ranking[i]->num_genes; j++) {
+            pt->debug_history[pt->debug_index++ % debug_size] = genes[j];
+        }
+    }
+}
+##else
+void dump_children(phenotype_t * pt) {}
+##endif /* GENETIC_DEBUG */
+
+static int __init genetic_lib_init(void)
+{
+    int retval;

```

```

+
+   kset_set_kset_s(&genetic_subsys, kernel_subsys);
+
+   retval = subsystem_register(&genetic_subsys);
+
+   return retval;
+}
+core_initcall(genetic_lib_init);
+
+static int __init genetic_boot_setup(char *str)
+{
+   if (strcmp(str, "on") == 0)
+       genetic_lib_enabled = 1;
+   else if (strcmp(str, "off") == 0)
+       genetic_lib_enabled = 0;
+
+   return 1;
+}
+
+static int __init genetic_mutation_rate_change_setup(char *str)
+{
+   int i;
+
+   if (get_option(&str,&i)) {
+
+       if (i > GENETIC_MAX_MUTATION_RATE)
+           i = GENETIC_MAX_MUTATION_RATE;
+       else if (i < 0)
+           i = 0;
+
+       mutation_rate_change = i;
+
+   }
+
+   return 1;
+}
+
+__setup("genetic=", genetic_boot_setup);
+__setup("genetic_mutate_rate=", genetic_mutation_rate_change_setup);
Index: linux-gf/lib/fingerprinting.c
=====
--- /dev/null
+++ linux-gf/lib/fingerprinting.c
@@ -0,0 +1,276 @@
+static int create_fingerprint(struct fingerprint * fp)
+{
+   int numerical_fp = 0;
+
+   numerical_fp |= fp->type;
+   numerical_fp <<= 1;
+
+   numerical_fp |= fp->pattern;
+   numerical_fp <<= 1;
+
+   numerical_fp |= fp->size;
+
+   return numerical_fp;
+}
+
+static long long get_top_fitness(phenotype_t * pt, struct fingerprint * fp)
+{
+   return pt->top_fitness[fp->type][fp->pattern][fp->size];
+}
+
+static int top_fitness_open(struct inode *inode, struct file *file)
+{
+   phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
+
+   return single_open(file, pt->ops->top_fitness_show, inode->u.generic_ip);
+}
+
+static struct file_operations top_fitness_ops = {
+   .open           = top_fitness_open,
+   .read           = seq_read,
+   .llseek        = seq_lseek,
+   .release       = single_release,
+};
+
+static int snapshot_open(struct inode *inode, struct file *file)
+{
+   phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
+
+   return single_open(file, pt->ops->snapshot_show, inode->u.generic_ip);
+}
+
+static struct file_operations snapshot_ops = {
+   .open           = snapshot_open,
+   .read           = seq_read,
+   .llseek        = seq_lseek,
+   .release       = single_release,
+};
+
+static int state_open(struct inode *inode, struct file *file)
+{
+   phenotype_t * pt = (phenotype_t *)inode->u.generic_ip;
+
+   return single_open(file, pt->ops->state_show, inode->u.generic_ip);
+}
+

```

```

+static struct file_operations state_ops = {
+    .open      = state_open,
+    .read      = seq_read,
+    .llseek    = seq_lseek,
+    .release   = single_release,
+};
+
+int genetic_init_fingerprinting(phenotype_t * pt)
+{
+    int i, j, k;
+    struct genetic_ops * ops = pt->ops;
+    int num_genes = pt->num_genes;
+
+    if (num_genes) {
+
+        pt->fp = (struct fingerprint *)kmalloc(
+            sizeof(struct fingerprint), GFP_KERNEL);
+
+        if (!pt->fp) {
+            printk(KERN_ERR "genetic_register_phenotype: not enough"
+                "memory\n");
+            return -ENOMEM;
+        }
+
+        reset_fp(pt->fp);
+
+        pt->fp_ss = (struct fp_snapshot *)kmalloc(
+            sizeof(struct fp_snapshot), GFP_KERNEL);
+
+        if (!pt->fp_ss) {
+            printk(KERN_ERR "genetic_register_phenotype: not enough"
+                "memory\n");
+            return -ENOMEM;
+        }
+
+        reset_fp_snapshot(pt->fp_ss);
+
+        pt->top_child = (unsigned long ***)kmalloc(
+            sizeof(unsigned long ***) * 2, GFP_KERNEL);
+
+        if (!pt->top_child) {
+            printk(KERN_ERR "genetic_register_phenotype: not enough"
+                "memory\n");
+            return -ENOMEM;
+        }
+
+        for (i = 0; i < 2; i++) {
+            pt->top_child[i] = (unsigned long **)kmalloc(
+                sizeof(unsigned long **) * 2,
+                GFP_KERNEL);
+
+            if (!pt->top_child[i]) {
+                printk(KERN_ERR "genetic_register_phenotype:\n"
+                    "not enough memory\n");
+                return -ENOMEM;
+            }
+
+            for (j = 0; j < 2; j++) {
+                pt->top_child[i][j] = (unsigned long *)kmalloc(
+                    sizeof(unsigned long *) * 2,
+                    GFP_KERNEL);
+
+                if (!pt->top_child[i][j]) {
+                    printk(KERN_ERR "genetic_register_phenotype: not enough memory\n");
+                    return -ENOMEM;
+                }
+
+                for (k = 0; k < 2; k++) {
+                    pt->top_child[i][j][k] = (unsigned long)ops->create_top_genes(pt);
+                    if (!pt->top_child[i][j][k])
+                        return -ENOMEM;
+                }
+            }
+        }
+    } /* if (num_genes) */
+
+    pt->top_fitness = (long long ***)kmalloc(sizeof(long long ***) * 2, GFP_KERNEL);
+    if (!pt->top_fitness) {
+        printk(KERN_ERR "genetic_register_phenotype: not enough"
+            "memory\n");
+        return -ENOMEM;
+    }
+
+    for (i = 0; i < 2; i++) {
+        pt->top_fitness[i] = (long long **)kmalloc(sizeof(long long **) * 2, GFP_KERNEL);
+        if (!pt->top_fitness[i]) {
+            printk(KERN_ERR "genetic_register_phenotype: not"
+                "enough memory\n");
+            return -ENOMEM;
+        }
+
+        for (j = 0; j < 2; j++) {
+            pt->top_fitness[i][j] = (long long *)kmalloc(
+                sizeof(long long *) * 2,
+                GFP_KERNEL);
+
+            if (!pt->top_fitness[i][j]) {
+                printk(KERN_ERR "genetic_register_phenotype: "
+                    "not enough memory\n");
+            }
+        }
+    }
+
+}

```

```

+         }
+         return -ENOMEM;
+     }
+     for (k = 0; k < 2; k++) {
+         pt->top_fitness[i][j][k] = 0;
+     }
+ }
+
+ pt->last_fingerprint = 0;
+
+ return 0;
+}
+
+static void decay_fitness(phenotype_t * pt, struct fingerprint * fp)
+{
+    long long fitness;
+    long dummy;
+
+    fitness = get_top_fitness(pt, fp);
+
+    /* reduce the fitness to eventually get new genes in */
+    fitness *= FP_DECAY;
+    divll(&fitness, 100, &dummy);
+
+    pt->top_fitness[fp->type][fp->pattern][fp->size] = fitness;
+}
+
+static void update_phenotype_top_performer(phenotype_t * pt, struct fingerprint * fp)
+{
+    long long top_fitness;
+    unsigned long * genes;
+    long long * avg_genes;
+    long dummy;
+    int i, j;
+
+    /* Decay the top fitness so not to have a fluke and have a
+     * high set which are less than optimal. So decay the top
+     * fitness so eventually these genes are phased out.
+     */
+    decay_fitness(pt, fp);
+
+    top_fitness = get_top_fitness(pt, fp);
+
+    if (pt->last_gen_avg_fitness >= top_fitness) {
+
+        pt->top_fitness[fp->type][fp->pattern][fp->size] = pt->last_gen_avg_fitness;
+
+        /* We don't need to track this if there's no genes! */
+        if (!pt->num_genes)
+            return;
+
+        avg_genes = (long long *)kmalloc(sizeof(long long) * pt->num_genes, GFP_KERNEL);
+        if (!avg_genes) {
+            printk(KERN_ERR "update_top_performers: unable to alloc space\n");
+            return;
+        }
+
+        memset(avg_genes, 0, sizeof(long long) * pt->num_genes);
+
+        for (i = 0; i < pt->num_genes; i++) {
+            for (j = 0; j < pt->num_children; j++) {
+                genes = pt->child_ranking[j]->genes;
+                avg_genes[i] += genes[i];
+            }
+        }
+
+        for (j = 0; j < pt->num_genes; j++)
+            divll(&avg_genes[j], pt->num_children, &dummy);
+
+        genes = (unsigned long *)pt->top_child[fp->type][fp->pattern][fp->size];
+        for (j = 0; j < pt->num_genes; j++)
+            genes[j] = avg_genes[j];
+
+        kfree(avg_genes);
+    }
+}
+
+static void update_top_performers(phenotype_t * master)
+{
+    phenotype_t * pt;
+    struct list_head * entry;
+    struct kobject * k = &master->kobj;
+
+    list_for_each(entry, &k->kset->list) {
+        pt = to_phenotype(to_kobj(entry));
+
+        if (master->uid & pt->uid && master->uid != pt->uid) {
+            update_phenotype_top_performer(pt, master->fp);
+        }
+    }
+
+    update_phenotype_top_performer(master, master->fp);
+}
+
+static void reintroduce_genes(phenotype_t * master)
+{
+    struct fingerprint * fp = master->fp;
+    phenotype_t * pt;
+    unsigned long * top_genes;

```



```

+static inline void divl(int32_t high, int32_t low,
+                       int32_t div,
+                       int32_t *q, int32_t *r)
+{
+    int64_t n = (u_int64_t)high << 32 | low;
+    int64_t d = (u_int64_t)div << 31;
+    int32_t q1 = 0;
+    int c = 32;
+    while (n > 0xffffffff) {
+        q1 <<= 1;
+        if (n >= d) {
+            n -= d;
+            q1 |= 1;
+        }
+        d >>= 1;
+        c--;
+    }
+    q1 <<= c;
+    if (n) {
+        low = n;
+        *q = q1 | (low / div);
+        *r = low % div;
+    } else {
+        *r = 0;
+        *q = q1;
+    }
+    return;
+}
+
+static inline void divll(long long *n, long div, long *rem)
+{
+    int32_t low, high;
+    low = *n & 0xffffffff;
+    high = *n >> 32;
+    if (high) {
+        int32_t high1 = high % div;
+        int32_t low1 = low;
+        high /= div;
+        divl(high1, low1, div, &low, (int32_t *)rem);
+        *n = (int64_t)high << 32 | low;
+    } else {
+        *n = low / div;
+        *rem = low % div;
+    }
+}
+
+##endif
+
+##endif /* #ifndef divll */
+
+##endif /* __LINUX_FINGERPRINTINT_H */
Index: linux-gli/include/linux/genetic.h
=====
--- /dev/null
+++ linux-gli/include/linux/genetic.h
@@ -0,0 +1,351 @@
+##ifndef __LINUX_GENETIC_H
+##define __LINUX_GENETIC_H
+/*
+ * include/linux/genetic.h
+ *
+ * Jake Moilanen <moilanen@austin.ibm.com>
+ * Copyright (C) 2004 IBM
+ *
+ * Genetic algorithm library
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2 of the GNU General Public License as published
+ * by the Free Software Foundation.
+ */
+
+##include <linux/kobject.h>
+##include <linux/list.h>
+##include <linux/timer.h>
+##include <linux/init.h>
+##include <linux/seq_file.h>
+##include <linux/kobject.h>
+
+##define GENETIC_HISTORY_SIZE          0x8
+##define GENETIC_HISTORY_MASK         (GENETIC_HISTORY_SIZE - 1)
+
+/* percentage of total number genes to mutate */
+##define GENETIC_DEFAULT_MUTATION_RATE 15
+
+/* XXX TODO Make this an adjustable runtime variable */
+/* Percentage that an iteration can jump within the range */
+##define GENETIC_ITERATIVE_MUTATION_RANGE 20
+
+/* the rate that GENETIC_DEFAULT_MUTATION_RATE itself can change */
+##define GENETIC_DEFAULT_MUTATION_RATE_CHANGE 4
+##define GENETIC_MAX_MUTATION_RATE    45
+##define GENETIC_MIN_MUTATION_RATE    10
+
+##define GENETIC_DEBUG                  0
+
+##ifdef CONFIG_FINGERPRINTING
+##define FP_DECAY                        90
+##define GENETIC_NUM_DEBUG_POINTS       5
+##else
+##define GENETIC_NUM_DEBUG_POINTS       4
+##endif

```

```

+
+#define GENETIC_PRINT_DEBUG          0
+#define gen_dbg(format, arg...) do { if (GENETIC_PRINT_DEBUG) printk(KERN_EMERG __FILE__ " : " format "\n" , ## arg); } while (0)
+#define gen_trc(format, arg...) do { if (GENETIC_PRINT_DEBUG) printk(KERN_EMERG __FILE__ " :%s:%d\n" , __FUNCTION__, __LINE__); } while (0)
+
+#define to_kobj(_entry) (struct kobject *)container_of(_entry, struct kobject, entry)
+
+struct gene_param_s;
+struct genetic_s;
+struct phenotype_s;
+
+struct genetic_child_s {
+    struct list_head    list;
+    long long           fitness;
+    unsigned long       num_genes;
+    void                *genes;
+    struct gene_param_s *gene_param;
+    void                *stats_snapshot;
+    int                 id;
+};
+
+typedef struct genetic_child_s genetic_child_t;
+
+#define to_gene_obj(obj) container_of(obj, struct gene_obj, kobj)
+#define to_gene_param_attr(_attr) container_of(_attr, struct gene_obj_attribute, attr)
+
+struct gene_obj {
+    struct kobject kobj;
+    struct genetic_s *genetic;
+    struct gene_param_s *gene_param;
+};
+
+struct gene_obj_attribute {
+    struct attribute attr;
+    ssize_t (*show)(struct gene_obj *, char *);
+    ssize_t (*store)(struct gene_obj *, const char *, size_t);
+};
+
+/* Here's a generic idea of what it the genes could look like */
+struct gene_param_s {
+    char *    name;
+    unsigned long min;
+    unsigned long max;
+    unsigned long initial;
+    void      (*mutate_gene)(genetic_child_t *, unsigned long);
+};
+
+typedef struct gene_param_s gene_param_t;
+
+#define GENE_PARAM_ATTR(_name, _mode, _show, _store) \
+struct gene_obj_attribute gene_param_attr_##_name = { \
+    .attr = { .name = __stringify(_name) , .mode = _mode }, \
+    .show = _show, \
+    .store = _store, \
+};
+
+extern struct kobj_type gene_param_ktype;
+
+#define to_phenotype_genetic(pt) (to_genetic(to_kset(pt->kobj.parent)->kobj.parent))
+#define to_phenotype(obj) container_of(obj, struct phenotype_s, kobj)
+#define to_phenotype_attr(_attr) container_of(_attr, struct phenotype_attribute, attr)
+
+struct phenotype_attribute {
+    struct attribute attr;
+    ssize_t (*show)(struct phenotype_s *, char *);
+    ssize_t (*store)(struct phenotype_s *, const char *, size_t);
+};
+
+struct phenotype_s {
+    struct kobject kobj;
+
+    struct list_head children_queue[2];
+    struct list_head *run_queue;
+    struct list_head *finished_queue;
+
+    struct genetic_ops *ops;
+
+    unsigned long num_children; /* Must be power of 2 */
+    unsigned long natural_selection_cutoff; /* How many children
+                                           * will survive
+                                           */
+
+    void unsigned long *stats_snapshot;
+    child_number;
+
+    struct kset genes;
+
+    /* percentage of total number of genes to mutate */
+    long mutation_rate;
+    unsigned long num_mutations;
+    unsigned long num_genes;
+
+    genetic_child_t **child_ranking;
+
+    void (*natural_selection)(struct phenotype_s *);
+
+    /* This UID is bitmap comprised of other phenotypes that contribute
+       to the genes */
+    unsigned long uid;

```

```

+
+     /* performance metrics */
+     long long          avg_fitness;
+     long long          last_gen_avg_fitness;
+
+     unsigned long      fitness_history_index;
+     long long          fitness_history[GENETIC_HISTORY_SIZE];
+
+ #if GENETIC_DEBUG
+     unsigned long      debug_size;          /* number of longs in
+                                           debug history */
+     unsigned long      debug_index;
+     long long          *debug_history;
+ #endif
+ #ifdef CONFIG_FINGERPRINTING
+     struct dentry       *fp_dir;
+     struct fingerprint *fp;
+     struct fp_snapshot *fp_ss;
+     unsigned long      ***top_child;
+     long long           ***top_fitness;
+     int                 last_fingerprint;
+ #else
+     long long           top_fitness;
+ #endif
+
+     long long           from_top;
+
+ };
+
+ typedef struct phenotype_s phenotype_t;
+
+ /**
+  * struct genetic_s - contains all data structures for a genetic plugin
+  * @name: string that will identify this genetic alg. in debugfs and printk
+  * @phenotype: list of all registered phenotypes
+  * @child_number: the running child index (< @num_children)
+  * @child_life_time: time in ms each child is ran before being swapped
+  * @num_children: number of children in each generation (must be a power of 2)
+  * @generation_number: increased once every child in a generation is ran
+  * @defaults: when 1 the genetic library will hold all genes at defaults
+  * @fingerprinting: when 1 the genetic library will use gene fingerprinting if CONFIG_FINGERPRINTING
+  */
+ struct genetic_s {
+     struct kobject      kobj;
+
+     struct kset          phenotypes;
+
+     struct timer_list   timer;
+
+     unsigned long       child_number;
+     unsigned long       child_life_time;
+     unsigned long       num_children;
+
+     unsigned long       generation_number;
+     int                 enabled;
+
+     /* private */
+ #ifdef CONFIG_FINGERPRINTING
+     int                 fingerprinting;
+ #endif
+ };
+
+ struct genetic_attribute {
+     struct attribute     attr;
+     ssize_t (*show)(struct genetic_s *, char *);
+     ssize_t (*store)(struct genetic_s *, const char *, size_t);
+ };
+
+ typedef struct genetic_s genetic_t;
+
+ struct genetic_ops {
+     void (*create_child)(genetic_child_t *);
+     void (*set_child_genes)(void *);
+     void (*calc_fitness)(genetic_child_t *);
+     void (*combine_genes)(genetic_child_t *, genetic_child_t *,
+                            genetic_child_t *);
+     void (*mutate_child)(genetic_child_t *);
+     void (*calc_post_fitness)(phenotype_t *); /* Fitness routine used when
+                                           * need to take into account
+                                           * other phenotype fitness
+                                           * results after they ran
+                                           */
+
+     void (*take_snapshot)(phenotype_t *);
+     void (*shift_mutation_rate)(phenotype_t *);
+     int (*gene_show)(struct seq_file *, void *);
+ #ifdef CONFIG_FINGERPRINTING
+     void (*get_fingerprint)(phenotype_t *);
+     void (*update_fingerprint)(phenotype_t *);
+     void (*create_top_genes)(phenotype_t *);
+     int (*top_fitness_show)(struct seq_file *, void *);
+     int (*snapshot_show)(struct seq_file *, void *);
+     int (*state_show)(struct seq_file *, void *);
+ #endif
+ };
+
+ int genetic_gene_obj_create(struct gene_param_s *, genetic_t *,
+                             struct gene_obj_attribute *, struct kset *);
+
+ #define to_genetic(obj) container_of(obj, struct genetic_s, kobj)
+ #define to_genetic_attr(attr) container_of(attr, struct genetic_attribute, attr)

```

```

+
+ #define GENETIC_TUNABLE(_var, _name)          \
+ static int g_show_##_name(struct gene_obj *g, char *buf) \
+ {                                             \
+     return sprintf(buf, "%lu\n", _var);     \
+ }                                             \
+ static int g_store_##_name(struct gene_obj *g, const char *buf, size_t size) \
+ {                                             \
+     char *p = (char *)buf;                  \
+     unsigned long temp;                     \
+     temp = simple_strtoul(p, &p, 10);       \
+     g->genetic->enabled = 0;                 \
+     if (temp < g->gene_param->max && temp > g->gene_param->min) \
+         g->gene_param->initial = temp;      \
+     return size;                            \
+ }                                             \
+
+ #define GENETIC_TUNABLE_ATTR(_name)         \
+ {                                             \
+     .attr = { .name = "current", .mode = 0644}, \
+     .show = g_show_##_name,                 \
+     .store = g_store_##_name,               \
+ }                                             \
+
+ /* Setup routines */
+ int genetic_init(genetic_t ** in_genetic, unsigned long num_children,
+                 unsigned long child_life_time, int fingerprinting,
+                 char * name);
+ phenotype_t * genetic_register_phenotype(genetic_t * genetic, struct genetic_ops * ops,
+                                         unsigned long num_children, char * name,
+                                         unsigned long num_genes, unsigned long uid);
+ void genetic_start(genetic_t * genetic);
+
+ /* Generic helper functions */
+ void genetic_generic_mutate_child(genetic_child_t * child);
+ void genetic_generic_iterative_mutate_gene(genetic_child_t * child, unsigned long gene_num);
+ void genetic_generic_combine_genes(genetic_child_t * parent_a,
+                                   genetic_child_t * parent_b,
+                                   genetic_child_t * child);
+ void genetic_create_child_spread(genetic_child_t * child, unsigned long num_children);
+ void genetic_create_child_defaults(genetic_child_t * child);
+ void genetic_general_shift_mutation_rate(phenotype_t * in_pt);
+ int genetic_generic_gene_show(struct seq_file *s, void *unused);
+
+ /* XXX do this more intelligently */
+ #ifndef DIVLL_OP
+ #define DIVLL_OP
+ #if BITS_PER_LONG >= 64
+
+ static inline void divll(long long *n, long div, long *rem)
+ {
+     *rem = *n % div;
+     *n /= div;
+ }
+
+ #else
+
+ static inline void divl(int32_t high, int32_t low,
+                        int32_t div,
+                        int32_t *q, int32_t *r)
+ {
+     int64_t n = (u_int64_t)high << 32 | low;
+     int64_t d = (u_int64_t)div << 31;
+     int32_t ql = 0;
+     int c = 32;
+     while (n > 0xffffffff) {
+         ql <<= 1;
+         if (n >= d) {
+             n -= d;
+             ql |= 1;
+         }
+         d >>= 1;
+         c--;
+     }
+     ql <<= c;
+     if (n) {
+         low = n;
+         *q = ql | (low / div);
+         *r = low % div;
+     } else {
+         *r = 0;
+         *q = ql;
+     }
+     return;
+ }
+
+ static inline void divll(long long *n, long div, long *rem)
+ {
+     int32_t low, high;
+     low = *n & 0xffffffff;
+     high = *n >> 32;
+     if (high) {
+         int32_t high1 = high % div;
+         int32_t low1 = low;
+         high /= div;
+         divl(high1, low1, div, &low, (int32_t *)rem);
+         *n = (int64_t)high << 32 | low;
+     } else {
+         *n = low / div;
+         *rem = low % div;
+     }
+ }

```

Sep 05, 06 0:16

genetic-lib.patch

Page 21/21

```
+      }  
+}  
+#endif  
+  
+#endif /* #ifndef divll */  
+  
+#endif
```

```

block: adds fingerprinting support to the block layer

The fingerprint is a formed from a triple of flags:

* read or write
* small or large
* random or sequential

Signed-off-by: Jake Moilanen <moilanen@austin.ibm.com>
Index: linux-rc/block/fingerprinting.c
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-rc/block/fingerprinting.c 2006-07-31 12:18:46.000000000 -0400
@@ -0,0 +1,205 @@
+/*
+ * block/fingerprinting.c
+ *
+ * Jake Moilanen <moilanen@austin.ibm.com>
+ * Copyright (C) 2006 IBM
+ *
+ * I/O Workload Fingerprinting
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2 of the GNU General Public License as published
+ * by the Free Software Foundation.
+ */
+/* TODOS:
+ * - Abstract so no so IO specific
+ * - Abstract types
+ */
+
+#include <linux/genhd.h>
+#include <linux/blkdev.h>
+#include <linux/fingerprinting.h>
+#include <linux/genetic.h>
+
+int fingerprint_state_show(struct seq_file *s, void *unused)
+{
+    phenotype_t * pt = (phenotype_t *)s->private;
+    struct fingerprint * fp = pt->fp;
+
+    if (fp->type == FP_TYPE_READ)
+        seq_printf(s, "read\t(%d)\n", FP_TYPE_READ);
+    else
+        seq_printf(s, "write\t(%d)\n", FP_TYPE_WRITE);
+
+    if (fp->pattern == FP_PATTERN_SEQ)
+        seq_printf(s, "sequential\t(%d)\n", FP_PATTERN_SEQ);
+    else
+        seq_printf(s, "random\t(%d)\n", FP_PATTERN_RAND);
+
+    if (fp->size == FP_SIZE_SMALL)
+        seq_printf(s, "small\t(%d)\n", FP_SIZE_SMALL);
+    else
+        seq_printf(s, "large\t(%d)\n", FP_SIZE_LARGE);
+
+    return 0;
+}
+
+int fingerprint_snapshot_show(struct seq_file *s, void *unused)
+{
+    phenotype_t * pt = (phenotype_t *)s->private;
+    struct fp_snapshot * ss = pt->fp_ss;
+
+    seq_printf(s, "read: %ld\n", ss->reads);
+    seq_printf(s, "write: %ld\n", ss->writes);
+
+    seq_printf(s, "avg_dist: %ld\n", ss->avg_dist);
+    seq_printf(s, "avg_size: %ld\n", ss->avg_size);
+
+    return 0;
+}
+
+int fingerprint_top_fitness_show(struct seq_file *s, void *unused)
+{
+    int i, j, k;
+    phenotype_t * pt = (phenotype_t *)s->private;
+
+    for (i = 0; i < 2; i++)
+        for (j = 0; j < 2; j++)
+            for (k = 0; k < 2; k++)
+                seq_printf(s, "top_fitness[%d][%d][%d]: %lld\n",
+                    i, j, k, pt->top_fitness[i][j][k]);
+
+    return 0;
+}
+
+/* This assumes that address matches up w/ head_pos */
+static void update_avg_dist(struct fp_snapshot * ss, long head_pos)
+{
+    long long tmp_dist;
+    unsigned long total_ops = ss->reads + ss->writes;
+    long dummy;
+
+    /* set it the first time through */
+    if (!ss->head_pos) {
+        ss->head_pos = head_pos;
+        return;
+    }

```

```

+     }
+     tmp_dist = ss->head_pos - head_pos;
+     if (tmp_dist < 0)
+         tmp_dist = -tmp_dist;
+
+     tmp_dist = tmp_dist - ss->avg_dist;
+
+     divll(&tmp_dist, total_ops, &dummy);
+     ss->avg_dist += tmp_dist;
+
+     ss->head_pos = head_pos;
+ }
+
+static void update_avg_size(struct fp_snapshot * ss, unsigned long size)
+{
+     unsigned long total_ops = ss->reads + ss->writes;
+     long long tmp_size;
+     long dummy;
+
+     tmp_size = size - ss->avg_size;
+     divll(&tmp_size, total_ops, &dummy);
+     ss->avg_size += tmp_size;
+ //     ss->avg_size += (size - ss->avg_size) / total_ops;
+ }
+
+void update_fp_snapshot(struct bio * bio)
+{
+     struct fp_snapshot * ss = bio->bi_bdev->bd_disk->fp_ss;
+
+     /* update type */
+     if (bio_data_dir(bio) == READ)
+         ss->reads++;
+     else
+         ss->writes++;
+
+     /* update pattern */
+ //     update_avg_dist(ss, bio_to_phys(bio));
+     update_avg_dist(ss, bio->bi_sector);
+
+     /* update size */
+ //     update_avg_size(ss, bio_iovec(bio)->bv_len);
+     update_avg_size(ss, bio_sectors(bio));
+ }
+
+/* Use this when there's multiple disks, and need to consolidate to a
+ * system wide fingerprint
+ */
+void consolidate_fp_snapshot(struct fp_snapshot * master, struct fp_snapshot * instance)
+{
+     unsigned long total_ops;
+     long dummy;
+     long long total_dist;
+     long long total_size;
+
+     BUG_ON(!master);
+     BUG_ON(!instance);
+
+     total_dist = master->avg_dist * (master->reads + master->writes);
+     total_size = master->avg_size * (master->reads + master->writes);
+
+     /* update operations */
+     master->reads += instance->reads;
+     master->writes += instance->writes;
+     total_ops = master->reads + master->writes;
+
+     /* update distance */
+     total_dist += (instance->avg_dist * (instance->reads + instance->writes));
+     if (total_ops) {
+         divll(&total_dist, total_ops, &dummy);
+         master->avg_dist = total_dist;
+     } else
+         master->avg_dist = 0;
+
+     /* update size */
+     total_size += (instance->avg_size * (instance->reads + instance->writes));
+     if (total_ops) {
+         divll(&total_size, total_ops, &dummy);
+         master->avg_size = total_size;
+     } else
+         master->avg_size = 0;
+ }
+
+void reset_fp_snapshot(struct fp_snapshot * ss)
+{
+     memset(ss, 0, sizeof(struct fp_snapshot));
+ }
+
+void reset_fp(struct fingerprint * fp)
+{
+     memset(fp, 0, sizeof(struct fingerprint));
+ }
+
+ //void calc_fp(struct fingerprint * fp, struct fp_snapshot * fp_ss, struct block_device * dev)
+void calc_fp(struct fingerprint * fp, struct fp_snapshot * fp_ss)
+{
+     /* type */
+     if (fp_ss->reads > (fp_ss->writes * FP_CLASS_READ_WRITE_RATIO))
+         fp->type = FP_TYPE_READ;
+ }

```

Sep 05, 06 0:16

io-fingerprinting.patch

Page 3/4

```

+     else
+         fp->type = FP_TYPE_WRITE;
+
+     /* pattern */
+//    if (fp_ss->avg_dist >= (block_size(dev) * FP_CLASS_PATTERN_RAND))
+    if (fp_ss->avg_dist >= (512 * FP_CLASS_PATTERN_RAND))
+        fp->pattern = FP_PATTERN_RAND;
+    else
+        fp->pattern = FP_PATTERN_SEQ;
+
+    /* size */
+    if (fp_ss->avg_size > FP_CLASS_SIZE_LARGE)
+        fp->size = FP_SIZE_LARGE;
+    else
+        fp->size = FP_SIZE_SMALL;
+}
+
+
+
+Index: linux-rc/block/ll_rw_blk.c
=====
--- linux-rc.orig/block/ll_rw_blk.c      2006-07-31 12:14:46.000000000 -0400
+++ linux-rc/block/ll_rw_blk.c          2006-07-31 12:18:46.000000000 -0400
@@ -28,6 +28,7 @@
 #include <linux/interrupt.h>
 #include <linux/cpu.h>
 #include <linux/blktrace_api.h>
+#include <linux/fingerprinting.h>
+
+/*
+ * for max sense size
@@ -2858,6 +2859,9 @@
     rw = bio_data_dir(bio);
     sync = bio_sync(bio);
+
+#ifdef CONFIG_FINGERPRINTING
+    update_fp_snapshot(bio);
+#endif
+
+/*
+ * low level driver can indicate that it wants pages above a
+ * certain limit bounced to low memory (ie for highmem, or even
+Index: linux-rc/include/linux/genhd.h
=====
--- linux-rc.orig/include/linux/genhd.h  2006-07-31 12:14:46.000000000 -0400
+++ linux-rc/include/linux/genhd.h       2006-07-31 12:18:46.000000000 -0400
@@ -60,6 +60,7 @@
 #include <linux/smp.h>
 #include <linux/string.h>
 #include <linux/fs.h>
+#include <linux/fingerprinting.h>
+
+struct partition {
+    unsigned char boot_ind;          /* 0x80 - active */
@@ -128,6 +129,7 @@
 #else
     struct disk_stats dkstats;
 #endif
+    struct fp_snapshot * fp_ss;
+};
+
+/* Structure for sysfs attributes on block devices */
+Index: linux-rc/block/genhd.c
=====
--- linux-rc.orig/block/genhd.c          2006-07-31 12:14:46.000000000 -0400
+++ linux-rc/block/genhd.c              2006-07-31 12:18:46.000000000 -0400
@@ -387,6 +387,20 @@
     jiffies_to_msecs(disk_stat_read(disk, io_ticks)),
     jiffies_to_msecs(disk_stat_read(disk, time_in_queue)));
 }
+static ssize_t disk_fp_read(struct gendisk * disk, char *page)
+{
+    return sprintf(page, "reads: %llx\n"
+        "writes: %llx\n"
+        "head_pos: %llx\n"
+        "avg_dist: %llx\n"
+        "avg_size: %llx\n",
+        (unsigned long long)disk->fp_ss->reads,
+        (unsigned long long)disk->fp_ss->writes,
+        (unsigned long long)disk->fp_ss->head_pos,
+        (unsigned long long)disk->fp_ss->avg_dist,
+        (unsigned long long)disk->fp_ss->avg_size);
+}
+
+static struct disk_attribute disk_attr_uevent = {
+    .attr = { .name = "uevent", .mode = S_IWUSR },
+    .store = disk_uevent_store
@@ -411,6 +425,10 @@
     .attr = { .name = "stat", .mode = S_IRUGO },
     .show = disk_stats_read
 };
+
+static struct disk_attribute disk_attr_fp = {
+    .attr = { .name = "fp", .mode = S_IRUGO },
+    .show = disk_fp_read
+};
+
+static struct attribute * default_attrs[] = {
+    &disk_attr_uevent.attr,
@@ -419,6 +437,7 @@
     &disk_attr_removable.attr,
     &disk_attr_size.attr,

```

Sep 05, 06 0:16

io-fingerprinting.patch

Page 4/4

```

+      &disk_attr_stat.attr,
+      &disk_attr_fp.attr,
+      NULL,
+    };
@@ -628,6 +647,10 @@
+      kobject_init(&disk->kobj);
+      rand_initialize_disk(disk);
+    }
+
+    disk->fp_ss = kmalloc(sizeof(struct fp_snapshot), GFP_KERNEL);
+    memset(disk->fp_ss, 0, sizeof(struct fp_snapshot));
+
+    return disk;
+  }
Index: linux-rc/block/Kconfig
=====
--- linux-rc.orig/block/Kconfig 2006-07-31 12:14:46.000000000 -0400
+++ linux-rc/block/Kconfig 2006-07-31 12:18:46.000000000 -0400
@@ -34,3 +34,9 @@
+   If unsure, say Y.
+
+   source block/Kconfig.iosched
+
+config FINGERPRINTING
+   bool "I/O Workload Fingerprinting"
+   help
+   Say Y here if you want workload data to be classified and
+   used to tune the I/O schedulers. Otherwise say N.
\ No newline at end of file
Index: linux-rc/block/Makefile
=====
--- linux-rc.orig/block/Makefile 2006-07-31 12:14:46.000000000 -0400
+++ linux-rc/block/Makefile 2006-07-31 12:18:46.000000000 -0400
@@ -10,3 +10,6 @@
+obj-$(CONFIG_IOSCHED_CFQ) += cfq-iosched.o
+
+obj-$(CONFIG_BLK_DEV_IO_TRACE) += blktrace.o
+
+obj-$(CONFIG_FINGERPRINTING) += fingerprinting.o

```

Sep 05, 06 0:16

autotest-genetic-ohone

Page 1/1

```
genes_dir = "/sys/kernel/genetic/ohone-cpuscheduler/phenotypes/general/genes/"
jobprofilers.add('catprofile', ['/proc/meminfo', '/proc/uptime'], 'meminfo', 2)
jobprofilers.add('catprofile', ['/proc/schedstat'], 'schedstat', 5)

genes = os.listdir(genes_dir)

for f in genes:
    jobprofilers.add('catprofile', [genes_dir + f + "/current"], f, 8)

def run_genetic_tests(tag):
    job.runtest(tag, 'kernbench', 2, int(2.5 * count_cpus()))
    job.runtest(tag, 'unixbench', 1, 'spawn shell dhry2reg')
    job.runtest(tag, 'interbench', 0)
    #job.runtest(tag, 'lmbench')
    #job.runtest(tag, 'reaim')
    #job.runtest(tag, 'tbench')

def genetic(i):
    os.system("echo %i > /sys/kernel/genetic/ohone-cpuscheduler/enabled" % i)
    return str(i)

run_genetic_tests(genetic(1))
run_genetic_tests(genetic(0))
```

Sep 05, 06 23:47

logfile

Page 1/4

commit 5eb6c2dfd43321769e7aff5a91df1c4da88a4f6c
 Author: root <root@dev14-sabine-lpl.ltc.austin.ibm.com>
 Date: Tue Aug 29 14:07:41 2006 -0400

Add some simple features to the test suite

commit a4cafb3ba90f8b4d77106c1ca96669b06275baee
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Wed Aug 23 10:08:49 2006 -0500

autotest http://test.kernel.org/autotest/ control file for testing genetic cpu sched

commit 79f532c88f730d53280757880fc1339aedbbcb3f
 Author: Brandon Philips <philips@plankton.ifup.org>
 Date: Mon Aug 21 00:11:37 2006 -0500

Modifying of ohone genes works without disturbing or breaking as. Perfect. Also fixed a fingerprinting bug (instant oops)

commit 1c527832e7679e41df70031582fdc4cb71138455
 Author: Brandon Philips <philips@plankton.ifup.org>
 Date: Sat Aug 19 17:28:16 2006 -0500

Kconfig moves to make the defaults for all of the options be y

commit 49713956cf1def5dad1535451a9e76399860a410
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Wed Aug 16 17:03:09 2006 -0500

Added enable/disable to sysfs for the plugins. Cleaned up attributes.

commit d4a0175795d28f21fa4d572a035c1048a07125c5
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Tue Aug 15 17:07:16 2006 -0500

sysfs support for genes works! (Doesn't crash immediatly, greatly simplified API)

commit 7ef45441ecccc303c78457433c01e72fbd7a7933
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Mon Aug 14 17:16:31 2006 -0500

Genes are now in the sysfs file system. Some genes now going to zero, need to investigate.

commit 063b5d662d81d0c1a34e2755468438db25d59719e
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Fri Aug 11 17:27:33 2006 -0500

First shot at adding sysfs attributes to gene_param_s. Crashes.

commit 27b8127195f867019123ee663467a25f8be09f48
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Tue Aug 8 13:05:24 2006 -0500

added phenotypes long long properties

commit 2001465532ef9a7c1b0e5963b5601ba3d21a4c2d
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Mon Aug 7 18:01:22 2006 -0500

Refresh for 2.6.18-rc4

commit 5a20e143baba4f94fd5557e29f55266a147eac64
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Mon Aug 7 17:56:18 2006 -0500

Added general properties to be exported to sysfs

commit 1f5abf3ef1463f228c3119e95a01682845aab7dd
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Mon Aug 7 16:48:56 2006 -0500

successful first move to sysfs and file name moves

commit dcc71dac2a9a32ab8acc9c7464c6eb1870e69e32
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Fri Aug 4 19:39:11 2006 -0500

It works with the phenotypes parent being the genetic_s, but not the phenotype kset. Grrrr

commit f80d9500bca0d443e087f5c3b8e7bd555837aaf6
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Fri Aug 4 15:37:57 2006 -0500

Consolidating patches, fixing up brokenness, oopses kernel on first generation switch

commit a3c27ba1dd6b7d86e1a4ffd03a23921265dd89e6
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Thu Aug 3 20:08:51 2006 -0500

sysfs first try

commit cee862a77474ccb608942838dca73595678f789d
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Mon Jul 31 19:42:52 2006 -0500

Added headers and cleanups for a code review

commit dd39e526039c875c3cf705b091ed1ca2c379281f
 Author: Brandon Philips <philips@plankton.ifup.org>
 Date: Mon Jul 31 13:01:08 2006 -0400

Tuesday September 05, 2006

1/4

Sep 05, 06 23:47

logfile

Page 2/4

Added headers and refreshed for 2.6.18-rc*

```
commit c3804981e9626212068d9d4c14fb495fbc27414c
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Fri Jul 14 16:47:33 2006 -0500
```

Changing licenses from GPL 2+ -> GPL 2

```
commit f956e115c89b61f9cc679342beb9ddecfd979fbd
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Fri Jul 14 16:23:52 2006 -0500
```

Moving the patches into sections to ease management

```
commit 1cae0d2d7ff7a44a7ee734381aa22c9d5d8ae8a2
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Fri Jul 14 16:20:27 2006 -0500
```

Breaking up the patches so they build at every patch level.

```
commit f8a76373f3736782f7f26020c9a14db1fcfb172
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Fri Jul 14 15:49:09 2006 -0500
```

Fingerprinting debugfs

```
commit 55389857bc97783ced23df6f195c5ca0856592
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Fri Jul 14 12:20:26 2006 -0500
```

Fingerprinting fixes for x86 and the latest patchset
 * Various type errors fixed
 * genetic_init changed to add a new parameter for fingerprinting

```
commit aa0b1d05537f2437b1dcf9e4871b2d95ac8f0562
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Thu Jul 13 19:07:47 2006 -0500
```

Disable VERBOSE debugging

```
commit c59b1dad8ee34d05ae72ebdbaf5322b7004017e
Merge: 2ba4acf... a773be4...
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Thu Jul 13 18:29:32 2006 -0500
```

Merge branch 'origin'

Conflicts:

genetic-as-sched.patch

```
commit 2ba4acf3f5783ba9b4f726efdf5ffabdafeecc88
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Thu Jul 13 18:15:23 2006 -0500
```

unsigned long -> long long for disk fitness statistics

```
commit a773be495e18d8b271b21955a7d3ba59443fcbcf
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Thu Jul 13 13:42:43 2006 -0500
```

Tab fixes

```
commit f1241abc6e3940501b42fbd8cbdf0eec2b27de5e
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Thu Jul 13 12:21:33 2006 -0500
```

Softtab -> hard tabs

```
commit 82876507c6361c7fbd06fa53d38f258ef8a8c546
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Thu Jul 13 12:09:07 2006 -0500
```

Added gene_show callback. (Fixes soft lock?)

```
commit a07f21c76e4d5c69b9d75029e43428fd5eeae184
Author: Brandon Philips <philips@plankton.ifup.org>
Date:   Thu Jul 13 00:51:24 2006 -0500
```

Added defaults file in the debugfs dir. N = run genetic lib; Y = hold all genes at defaults

```
commit 4870cf150dfee11fee064e5b1fe08a4d4189b170
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Wed Jul 12 16:11:30 2006 -0500
```

Investigations into x86 soft locks
 * Added GENETIC_DEBUG_VERBOSE printk's
 * Printing error for phenotype genes

```
commit 44b58ac382100a7ac65ff8e16d73aef92c254588
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Mon Jul 10 19:32:13 2006 -0500
```

Additional debugfs entry definitions.

```
commit cae8699ab12c3a2a3bc756f5807e0e77b836256d
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date:   Fri Jul 7 17:41:58 2006 -0500
```

Sep 05, 06 23:47

logfile

Page 3/4

Minor fixes that were causing warning on x86

commit 28d31757da824ca08826b99d68cd14ce7031974c
 Author: Brandon Philips <philips@karen.austin.ibm.com>
 Date: Fri Jul 7 17:41:27 2006 -0500

Do `_NOT_ negate unsigned values... ever`

* This fixes the copy paste bug that caused failure on x86 and UML x86

commit df6c988811f5089676d4dcf692adc46abb2a6e1c
 Author: Brandon Philips <philips@plankton.ifup.org>
 Date: Fri Jul 7 10:07:31 2006 -0500

Added `uml host target`

commit b997a2f1b2c8e33f9a9cfa5140c0bef295c46dbe
 Author: Brandon Philips <brandon@ifup.org>
 Date: Thu Jul 6 18:00:23 2006 -0500

Initial work to move from `proc` -> `debugfs`

Next up: `genetic-lib` on off switch

commit ced1ef52fab304e7b48af1931227328ce8ebbc0
 Author: Brandon Philips <brandon@ifup.org>
 Date: Wed Jul 5 17:20:54 2006 -0500

Added `Kconfig` options for the `genetic cpu scheduler` (`i386` and `PPC`)

commit a2d420835d2702b562776b1ddfb9cd34f7f8f85d
 Author: Brandon Philips <brandon@ifup.org>
 Date: Wed Jul 5 15:50:56 2006 -0500

renamed: `genetic-cpu-pre-phenotypes.patch` -> `genetic-cpu-sched.patch`

commit 57fcaac11e573cf078ed7c62149bbcc76bad3046
 Author: Brandon Philips <brandon@ifup.org>
 Date: Wed Jul 5 15:48:10 2006 -0500

Changed the comment about unsigned long's to make more sense

commit 31c8191212e2b05f4f0c4c26fcb6e6c7088cd200
 Author: Brandon Philips <brandon@ifup.org>
 Date: Wed Jul 5 15:29:40 2006 -0500

Added kernel doc to important functions and cleaned up to kernel standards

commit ba415db1955579c9646029468e1539549aaa123c
 Author: Brandon Philips <brandon@ifup.org>
 Date: Wed Jul 5 14:51:43 2006 -0500

Disable `GENETIC_DEBUG` and unexpand some soft tabs

commit fb31e11da2a7af196b6dae2c0a68393d80a7d199
 Author: Brandon Philips <brandon@ifup.org>
 Date: Wed Jul 5 14:43:08 2006 -0500

All the genes are running. The bug was tracked down by Jake Moilanen.

* genes are now unsigned long instead of unsigned int to comply w/ `gl generic casts`

commit aedb42fa9bd433ab11866f766d819f50a482e7da
 Author: Brandon Philips <brandon@ifup.org>
 Date: Fri Jun 30 15:05:06 2006 -0500

Fixed a bug with disabling `GENETIC_DEBUG`, increased gene count to 8

Known `O(1)-gl` issues

* Only 8 of the possible 12 genes in use

Known `gl` Issues

* A gene count above 8 for a phenotype causes an access error in `genetic_switch_child`

commit 8cc88b4a1394abb751d5687d1844954af9f1d5b9
 Author: Brandon Philips <brandon@ifup.org>
 Date: Fri Jun 30 10:06:38 2006 -0500

It boots and runs.

What works

* It boots
 * Debugging stats are printed

Known Issues

* Two genes are being used, the rest are disabled: `child_penalty`, `parent_penalty`
 * Need to investigate which gene is causing `softirq` crash (`kthread` issue?)
 * Negative fitness for latency and context switch

commit a4f698d2cd9a8538880d1a3c7dab8c2166a9bdfc
 Author: Brandon Philips <brandon@ifup.org>
 Date: Thu Jun 29 17:05:20 2006 -0500

I don't know what the change is

commit 8e113369b0a4f28f746ae5c67b093422279e0891
 Author: Brandon Philips <brandon@ifup.org>
 Date: Thu Jun 29 16:11:35 2006 -0500

Tuesday September 05, 2006

3/4

Sep 05, 06 23:47

logfile

Page 4/4

create_child fixes

commit c5ea26b4557c43224685525224caa39a10e2d21a
Author: Brandon Philips <brandon@ifup.org>
Date: Thu Jun 29 14:47:16 2006 -0500

Move genetic init to a core_initcall

commit de53b80e334887822d5b48fc804277ab35e3c8f8
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date: Thu Jun 29 14:26:48 2006 -0500

It compiles, now for the testing

commit eel4607d63a635042e69fed2c7838f7c47belfee
Author: Brandon Philips <philips@plankton.ifup.org>
Date: Thu Jun 29 08:32:48 2006 -0500

Additional porting to the O(1) scheduler; it almost compiles

commit 5fed1fe203fb86a2ffb2d1c40f3b8b13eb0ec5b1
Author: Brandon Philips <philips@karen.austin.ibm.com>
Date: Wed Jun 28 19:53:49 2006 -0500

Initial commit