Wonderful : A Terrific Application and Fascinating Paper

Your N. Here
Your Institution

Second Name
Second Institution

Abstract

Your Abstract Text Goes Here. Just a few facts. Whet our appetites.
TeX into TeXnicians and TeXperts.
http://www.scu.edu/˜bush
http://www.pku.edu/∼bush
mother-in-law
pages 1–12
yes —or no
0, 1 and -1
$1-2$
Δ

1 Introduction

A paragraph of text goes here. Lots of text. Plenty of interesting text.

More fascinating text. Features[1] galore, plethora of promises.

2 This is Another Section

Some embedded literal typset code might look like the following :

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int result;
    int arg0;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
```

Figure 1: Wonderful Flowchart

```
    return TCL_OK;
}
```

Now we're going to cite somebody. Watch for the cite tag. Here it comes [**?**, **?**]. The tilde character (˜) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

3 This Section has SubSections

3.1 First SubSection

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

3.2 New Subsection

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{, %}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

`_1008e614_Vector_p`

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

```
size_t :  fread ptr size nobj stream
```

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

3.3 How to Build Your Paper

You have to run `latex` once to prepare your references for munging. Then run `bibtex` to build your bibliography metadata. Then run `latex` twice to ensure all references have been resolved. If your source file is called `usenixTemplate.tex` and your `bibtex` file is called `usenixTemplate.bib`, here's what you do:

```
latex usenixTemplate
bibtex usenixTemplate
latex usenixTemplate
latex usenixTemplate
```

3.4 Last SubSection

Well, it's getting boring isn't it. This is the last subsection before we wrap it up.

4 Evaluation

We have implemented the proposed novel algorithm which is described in previous sections. The algorithm requires no more than 400 SLoC added to Linux while only around 200 SLoc added to Xen hypervisor. As mentioned before, the new code added to Linux kernel is to build a cache pool for guest page tables and a slight modification to Xen hypervisor is to avoid unnecessary IOTLB flushes.

This section evaluates the I/O performance as well as CPU usage of our implementation by running both macro- and micro-benchmark kits.

4.1 Experimental Setup

Our experimental platform is a LENOVO QiTianM4390 PC with Intel Core i5-3470 running at 3.20 GHz, four CPU cores available to the system. We enable VT-d feature in the BIOS menu, which supports page-selective invalidation and queue-based invalidation interface. Xen version 4.2.1 is used as the hypervisor while domain0 uses the Ubuntu version 12.04 and kernel version 3.2.0-rc1. In addition, domain 0 should configure its grub.conf to turn on I/O address translation for itself and also print log information to a serial port in debug mode.

4.2 Micro-Benchmarks

To evaluate our algorithm, a possible micro-way is to measure the frequencies of IOTLB-flush with/without the cache pool. Thus, by making use of a tiny stress tiny tool that we develop, the measurements are performed under three particular settings.

The first setting is a normal control group where system is only running the tool with cache pool disabled. Specifically, the tool aims to give rise to frequent updates of page tables by launching a default browser(e.g., Mozilla Firefox 31.0 in the experiment), opening new tab one by one and closing the browser gracefully. As for the interval time between browser invocation and closure, we set it to five minutes since the frequency of IOTLB-flush in the control group will become relatively stable in five minutes every time the tool is launched.

Then the cache pool is enabled and enjoys different pressure imposed by the tool under the rest two settings. In one setting called low-pressure group, the system activates the cache pool mechanism through a kernel loadable module and runs the tool concurrently.

While in the other setting called high-pressure group, the system activates the cache pool mechanism five minutes (to match the interval time) after the tool begins running. Consequently, the high pressure lies in creating/terminating a large number of Firefox processes in five minutes rather than using too much memory or CPU cycles, while the system is idle in the low-pressure group when activating the mechanism without any update of page tables occurred.

In addition, please note that under all settings the time that system starts to run the tool is ten minutes after system boots up when the log information from serial port indicates that the system do not cause IOTLB-flush any more. Specifically, We have been aware that a type change of a page table will invoke the function iotlb_flush_qi(), and then an corresponding IOTLB entry will be invalidated in a page granularity. Thus, a global counter is set to record invocation times of that function and an average number per minute is calculated, representing a frequency of IOTLB-flush per minute. Thats the reason why the average number logged drops to zero, meaning that IOTLB does not be flushed any more.

Now, we can take a look at figures in the three different groups.Y-axis in figure 1 represents the frequency of IOTLB-flush, corresponding to the time interval (i.e., one minute) within thirty minutes in x-axis. As mentioned before, the time to start in x-axis is when the tool is launched, causing frequent invocations of iotlb_flush_qi(). From this figure, the frequency in control group remains quite stable and high after five minutes. By contrast, frequency in low-pressure group drops to zero in a very short time and remains stable from then on, while frequency in high-pressure group has very similar behavior with that of control group in the first five minutes, but is reduced to zero quickly due to the cache pool.

It can be concluded that once our proposed algorithm is enabled, the IOTLB frequency can be reduced to zero, which we can regard it as a maximum reduction.

Having discussed about the I/O performance in a micro way, now lets move to the CPU time that each group will take up when the tool is put to use. Specifically, the time to allocate and free a particular level of page table (e.g., page global directory) from the cache pool or from the buddy system is calculated to evaluate the effects that the algorithm has on CPU usage. From figure 2, compared with control group which interacts with the buddy system, low-pressure group takes up much less CPU time to allocate and free pages from the pool while high-pressure group also decreases to a similar level five minutes after the cache is turned on. And so do the rest two levels of page tables.

To further illustrate the algorithms advantage in CPU usage, Lmbench is used to measure time that process-related system commands cost (i.e., fork+exit, fork+execve, fork+/bin/sh -c), shown in figures xxx to xxx. The configuration parameters are selected by default, except parameters of processor MHz, a range of memory and mail result, since CPU mhz of our test machine is 3.2GHz rather than the default one, memory range uses 1024MB to save time that Lmbench-run takes and we need no results mailed. From the first two figures, command of fork+exit in the group of tool-off+algorithm-on only costs xxx in microseconds, which is the shortest among the rest three groups, since only lmbench is running without the tool taking up CPU cycles while the algorithm is taking effect. And time costs of the other two commands in the rest figures also support that group of tool-off+algorithm-on costs a least amount of time. As a result, the algorithm has helped to shorten the time that creating/terminating a process costs and thus increase efficiency.

Besides the I/O efficiency and CPU usage, the algorithm should also be evaluated in the aspect of memory usage since three levels of cache pools have been built to support a fast process creation/termination. We try to reach a satisfying balance between time and space. From figure 5 to 7, low-pressure group takes up to only 250 pages(i.e., $1000\text{K} < 1\text{M}$) in the long time run while high-pressure group caches only 210 pages at most. It is reasonable that high-pressure group has less pages in the pool since a certain amount of page tables is freed to the buddy system right after the cache pool is put to use. And the proportion among the total pages of each level of page table including pages both in pool and in use also matches kernel paging mechanism. On top of that, when to free the pages from the pool to the buddy system is based on the training study. Specifically, we use PGD pool as an example. Pages in pool will not be freed unless two conditions are true. 1) a proportion between pages in pool and that in use is greater than $1:1$; 2) total number of pages in pool and in use exceeds 10 pages. As for the page number to free, the difference number between them is set in order to adjust the proportion to $1:1$. An equation below can clearly state this point: $\Delta num\_to\_free = num\_in\_pool - num\_in\_use$.

Note that the method to determine the proportion and total page number is based on an experiment when no pages are freed, which is the so-called training study. Further discussions about when to free will be talked about in the future work.

4.3 Micro-Benchmarks

Different micro-benchmark tests have reached a good result for the algorithm. And then we will make use of macro-benchmarks to evaluate its effect on overall system performance as well as I/O performance. Within each benchmark test, two experimental settings are designed for fair comparison. One is benchmark with the stress test tool, and the other has nothing but the benchmark. Based on each setting, performance of the algorithm is compared with that of general guest system without optimization of the algorithm. As a result, each benchmark will have four groups of experimental data, i.e., (tool-on+algorithm-on, tool-on+algorithm-off), (tool-off+algorithm-on, tool-on+algorithm-on).

SPECint_2006v1.2 is chosen to test what effects that the algorithm will have on the whole system performance. And 12 benchmarks of SPECint are all invoked with EXAMPLE-linux64-ia32-gcc43+.cfg for integer computation, results of which produce figures x.3.1 to x.3.4.

From figures x.3.1 to x.3.2, system is running SPECint as well as the test tool with/without the algorithm. Lets use 400.perlbench as an example to explain something. The amount of time in seconds that the benchmark takes to run with the algorithm is from xxx to xxx, and the ration range is from xxx to xxx, both of which are within ranges of perlbench without the algorithm. And this explanation also works for the rest 11 benchmarks. And so do the rest two figures. Thus, we think that experimental results of each benchmark are almost the same with/without the algorithm, which indicate that the algorithm does not have any bad effect on system performance.

As for I/O performance, we use netperf to evaluate the performance of network-intensive workloads. To overcome the adverse effect caused by real network jitter, the tested machine is connected directly to a tester machine by a network cable. Then we measure the network throughput from the tester machine being a client by sending a bulk of TCP packets to the tested machine being a server. More specifically, tester client connects to the tested server by building a single TCP connection, test type of which is TCP_STREAM, and test length of which lasts 60 seconds. On top of that, the TCP_STREAM test of netperf is run 30 times to obtain an average value of throughput.

As can be seen from figures xxx to xxx, the throughput range of group (tool-on+algorithm-on) is within that of group (tool-on+algorithm-off). And so do the rest two groups. Intuitively, the results indicate that the algorithm of IOTLB optimization has no contribution to the performance improvement, which contradicts with that of micro-benchmark.

Actually, Nadav Amit [xxx] demonstrates that the virtual I/O memory map and unmap operations consume more CPU cycles than that of the corresponding DMA transaction so that the IOTLB has not been observed to be a bottleneck under regular circumstances. Thus, only when the cost of frequent mapping and unmapping of IOMMU buffers is sufficiently reduced, the guest physical address resolution mechanism becomes the main bottleneck. Furthermore, he proposes the so-called pseudo pass-through mode and utilizes a high-speed I/O device (i.e., Intels I/O Acceleration Technology) to reduce time required by DMA map and unmap operations so that IOTLB becomes the dominant factor.

As a result, it is quite reasonable that the experimental results of netperf with/without the algorithm are almost the same.

5 Acknowledgments

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

6 Availability

It's great when this section says that MyWonderfulApp is free software, available via anonymous FTP from

```
ftp.site.dom/pub/myname/Wonderful
```

Also, it's even greater when you can write that information is also available on the Wonderful homepage at

```
http://www.site.dom/~myname/SWIG
```

Now we get serious and fill in those references. Remember you will have to run latex twice on the document in order to resolve those cite tags you met earlier. This is where they get resolved. We've preserved some real ones in addition to the template-speak. After the bibliography you are DONE.

Notes

[1] Remember to use endnotes, not footnotes!