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);
    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.

Figure 1: Wonderful Flowchart

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 performance of our implementation by running both micro- and macro-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 to print log information to a serial port in debug mode.

Besides that, measurements in both micro- and macro-benchmark kits are performed under two particular settings. The first setting is a normal control setting with the cache pool disabled while the other one is an experimental setting with cache enabled. And under both settings the time that system starts to activates the cache mechanism as well as launch those benchmarks is ten minutes after system boots up when the log information from serial port indicates that the system is in an idle state and do not cause IOTLB-flush any more. Note that the cache is disabled by default.

Specifically, We have been aware that a type change of a page table (e.g., from a page of Writable to a page of Page Global Directory) will invoke the function iotlb_flush_qi() to flush a corresponding IOTLB entry. And that is how page table updates affect IOTLB-flush. Thus, we put a global counter in the function body to record invocation times of the function and then an average counter per minute is calculated which is called a frequency of IOTLB-flush. When the logged average counter drops to zero, it means that IOTLB does not be flushed any more.

As a result, we utilize one micro-benchmark to evaluate the frequency of IOTLB-flush, CPU usage as well as memory usage while macro-benchmarks give an assessment on overall system performance.

4.2 Micro-Benchmarks

To begin with, a tiny micro-benchmark tool that we develop is introduced. Specifically, the tool launches a default browser(e.g., Mozilla Firefox 31.0 in the experiment), opens new tabs one by one and five minutes then closes the browser gracefully in an infinite loop, which aims to constantly creates/terminates a large number of Firefox processes and brings network-intensive workloads, thus giving rise to frequent updates of page tables, which then cause IOTLB to be flushed. As for the interval time of five minutes between browser invocation and closure, it is set because the frequency of IOTLB-flush in the control group will become relatively stable five minutes after the tool is launched.

As a result, system launches the tool with the cache disabled in the control setting. While in the experimental setting, we provide an interface for users to flexibly activate the cache mechanism through a kernel loadable module in order to evaluate the algorithm more precisely. More specifically, in a cache-dynamic-enabled group, the tool starts to run for a certain amount of time (e.g., five minutes) before cache is enabled in an on-demand way while the system tests the tool right after the cache is enabled in a cache-pre-enabled group. And this is how figures xxx are produced.

Now, we can take a look at the figure xxx. Y-axis in figure xxx represents the frequency of IOTLB-flush, corresponding to the time interval (i.e., one minute) of x-axis for the first thirty minutes that the running tool has taken up. From this figure, the frequency in the cache-disabled group remains quite stable and high after five minutes. By contrast, the frequency in the cache-pre-enabled group drops to zero in a very short time. As for the cache-dynamic-enabled group, the frequency has a very similar trend with that of the cache-disabled group in the first five minutes, but is reduced to zero due to the cache pool, which is just like that of the cache-pre-enabled group.

We can safely conclude that once our proposed algorithm is enabled, the IOTLB frequency can be reduced to zero quickly, which can be regarded as a maximum reduction.

Having discussed about the I/O performance in a micro way, now lets move to the CPU usage that each group will take up. Specifically, each level of page table has its allocation functions and free functions, e.g., pgd_alloc() and pgd_free() and the execution time that every related function is calculated per minute. As can be seen from figure xxx, the cache-pre-enabled group consumes 30% less CPU time when interacting with the pool, compared with that of the cache-disabled group which interacts with the buddy system. And the cache-dynamic-enabled group decreases to a similar level with that of the cache-pre-enabled group right after the cache is turned on.

Besides the I/O efficiency and CPU usage, the algorithm is 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 figures xxx to xxx, the cache-pre-enabled group takes up to only 250 pages(i.e., $(1000\text{K} = 250) < 1\text{M}$) in the long time run while the cache-dynamic-enabled group caches 210 pages at most. It is reasonable that the cache-dynamic-enabled group has less pages in the pool since a certain amount of page tables is freed to the buddy system before the cache pool is put to use. And only less than 1M memory is thus consumed in both groups. 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) a 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: $\Delta\text{num\_to\_free} = \text{num\_in\_pool} - \text{num\_in\_use}$. Note that the method to determine the proportion and the total page number is based on an experiment when no pages are freed, which is the so-called training study.

Since the training study that we rely on to free a page is dependent on a specific application, it does not work for other applications. We have a further discussion about when to free in the future work.

4.3 Macro-Benchmarks

Different micro tests have reached a good result for the algorithm, and then we will make use of macro-benchmarks to evaluate its effects on overall system performance. Each setting has a group for the benchmarks to run, i.e., a cache-disabled group and a cache-pre-enabled group.

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 xxx to xxx.

From figures xxx to xxx, system is running SPECint 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. Thus, we think

that the 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.

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 3.2 GHz rather than the default one, memory range uses 1024 MB to save time that Lmbench-run takes and we need no results mailed. From the figure xxx, command of fork+exit in the cache-pre-enabled group only costs xxx in microseconds, xxx% lee than that of the cache-disabled group, since the algorithm is taking effect. And the costs of the rest two commands in the figures xxx and xxx also support that the cache-pre-enabled group costs less. As a result, the algorithm has optimized the CPU usage when creating/terminating a process.

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 the cache-pre-enabled group is within that of the cache-disabled group. Intuitively, the results indicate that the algorithm of IOTLB optimization has no contribution to the performance improvement, which seemingly contradicts with the result from micro experiments.

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 netperf results 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!