;login:
OCTOBER 2013 VOL. 38, NO. 5
ELECTRONIC SUPPLEMENT

Conference Reports

HotPar '13: 5th USENIX Workshop on Hot Topics in Parallelism

San Jose, CA
June 24-25, 2013

Panel

Tools in the Real World

Summarized by Rik Farrow (rik@usenix.org)
Panelists: Niall Dalton, Calxeda; Brandon Lucia, University of Washington and Microsoft Research; Tipp Moseley, Google; Paul Peterson, Intel Corporation

Brandon Lucia has just gotten his Ph.D. from the University of Washington and is going next to MSR. Brandon started talking about software development tool research. Development tools eat data, such as programming traces and source code. Next, we need to abstract the data (for example, convert program traces to event traces). Abstractions helps us facilitate analysis, the final step, for example, in suggesting a solution for a problem in performance.

Brandon provided a concrete example from his own work, a project called Recon (recon.cs.washington.edu), for concurrency debugging. Recon uses CPU hardware to monitor shared data accesses, uses this to build context-aware graphs, and analyzes these graphs to reconstruct the root cause of a failure.

Brandon ended by covering some trends. Statistical modeling and analysis allows you to take big piles of data and make sense out of them, distilling the data into a model. The next trend is the collection of data in real time, such as instrumenting all of Google's servers to capture rare events in situ. Third, tools can also be used for automation, not just for analysis but also for fixing problems. The last trend Brandon talked about was closing the gap between hardware architecture and software tool designers. Hardware support allows you to collect data that you wouldn't otherwise be able to collect.

Tipp Moseley began by saying that tools solve problems. Google collects hundreds of thousands of profiles every day, including hardware counters (instructions per second, branch misprediction, cache misses) and software profiles (heap size, growth, lock contention, disk fragmentation). They process this data to produce reports on potentially anomalous results for applications, libraries, and even functions. Because Google owns the entire stack, every time you submit a change, your change includes tests so that the change can be analyzed. Tipp said that their tools work well for uncovering race cases, while some other tests, like load tests, are difficult to test. Google does profile applications in production, but scale is a huge problem. A one-in-a-million race condition will happen all the time at Google's scale. Static analysis works poorly at this scale, because the systems are so large with many interacting programs on distributed systems.

Tipp said that the really hard problems cross boundaries. For example, each Web request comes in through load balancers, to frontends, to backends, then to storage. It becomes very difficult to figure out where a problem occurs in this chain, discovering what causes long tail latency, for example, in performance.

Tipp wants tools that have low overhead, such as sampling that takes less than 3%, as well as more hardware counters.

Niall Dalton said the most important tool is coffee. Niall displayed a chart on which there is a latency spike every 500 ms after an OS upgrade, and asked how we would solve this. In another example, a new version of a system comes in, and again there are latency spikes that show up routinely, but software tools fail to discover what's causing the problem. Niall explained that the problem lay in the BIOS, and that he had to hack the BIOS to fix the problem. Both examples were single applications on dedicated systems. Niall next described having two applications on the same box, both stressing RAM access, but tools that trace applications wouldn't see that. Niall said changes to disk seek patterns, network incast, and the effects of big data applications that are not on the system under observation but affect its performance are like a "whale swimming by." So you have your own problems, plus your neighbors'.

A lot of us have built ad hoc tools over the years, but the hardest problem is to discover where, deep in the system, something is going on. Just think of dueling schedulers. And it might take 2,000 hours before a kernel crash occurs.

Paul Peterson said that when you are in the software tools business, your software will work better on your hardware than on other hardware. Paul added that he was speaking for himself, not Intel. Although people are most familiar with Intel as a hardware company, Intel has also been a software company that has been working in the world of parallelism since multicore CPUs became common. Intel works with BIOS, device drivers, operating systems for optimizing performance, and with 14,000 engineers worldwide.

Paul works on the Parallel Studio suite of products, focusing on the node level, but also on the cluster level, with tracing and analysis tools at each level. For example, Advisor XE helps people design and build parallel programs. They also have Composer, Intel MPI, VTune amplifier, and Inspector, which looks for memory leaks. Intel produces enabling software that helps developers.

The chair started off by saying that coffee is his favorite tool, too. He then asked Tipp whether some of the concerns he has are Google-only problems, that is, only large data companies have these problems. Tipp said that in 10 years, everyone is going to have to deal with them, even on smartphones. Another panelist said that cloud computing is already producing environments that look like a lot of problems within Google. Paul ranked his top three list of customer complaints: tools shouldn't break (especially debugging tools—broken debuggers really piss people off); speed matters (for anything other than hardware tools) and overhead should be less than 10%; and finally, the tool doesn't produce enough data. People want tools to be faster and richer. Niall said that these problems already exist, say, if they want traces on a group of systems instead of one.

Next, the chair asked Tipp what they did to solve race detection. Tipp said he didn't solve this himself, but that much smarter people built tools built on Valgrind that just seem to work. A lot of the work is based on fine-tuning edge cases. Google has good test coverage, but doesn't have good tools for doing race detection on code working at production scale. Brandon said that he didn't think that race detection problems are solved, that the overhead is too high (10x). Tipp said that he wanted that side, the production side, solved as well.

An audience member asked Tipp about publishing statistics and info that would help academics work on realistic projects. Tipp said that Google often does publish measurement papers. Being in academia and working on these problems is tough, but Google does try to. Google has both student and professor internships that can help provide real-world experience with the unexpected things that will happen. Another panelist pointed out that working within a cloud is similar to not having access to traces from within Google, because cloud providers hide most infrastructure details. So people should try and build tools that can work in these environments and uncover performance issues, like long tails.

Paul pointed out that Intel funds research groups. He said that Kim Hazelwood (Google) is an example of that. While at the University of Virginia, she worked on binary instrumentation (PIN), which Intel funded. Intel is looking for proposals, especially if there's a hardware hook to it. If you have a hardware idea, you've chosen the highest bar to cross, but the one with the highest payoff. If you want to measure something constantly with high performance, you want to use hardware.

Brandon said that the cloud was a useful tool for scaling out; although it's not the same as Google-scale, it is useful. Niall suggested building a microserver, using low-powered systems, and fake up a half-rack that can provide a nasty set of problems. Someone asked for a good set of software to run. Niall said there is no good single answer, but you should start with Linux, then throw in noise generators, latency-sensitive stuff, bandwidth hogs, and just abuse your machine because that's what users in the real world are going to do.

Another person said he had presented a tool at SOSP for debugging race conditions, but ran into problems that occurred in the libraries layers, not in the software under test. Paul replied that if glibc is using a particular lock, you must be able to communicate that to the tool running the test. Paul is now looking at problems that comes from garbage collection, because data is getting moved around, so many more people may start having similar problems. The questioner then moved away from the mike but kept talking, and the rest of his side of the discussion was lost.

Keynote Address

Parallelism in the Cloud

Eric Brewer, University of California, Berkeley, and Google
Summarized by Rik Farrow (rik@usenix.org)

Eric Brewer began by saying we have had giant-scale services for 17 years now, using a three-tiered architecture: with highly available, load balanced, frontend servers; stateless workers that can easily be restarted; and durable storage that is replicated and highly available. Latency of Web services matters a lot, with various claims for lost revenue based on customers who are delayed by additional hundreds of milliseconds. Eric displayed Jeff Dean's performance slide, showing the three tiers again, but with the second tier largely replaced with cache servers designed to reduce latency, but often hurting tail latency.

Eric said that caching, and prediction in general, can hurt tail latency. Other things that can hurt tail latency include parallelism, where the response is limited by the slowest reply; virtualization, because of both scheduling issues and hiding of real resources; and logs, where write can be fast but occasional compactions cause blips in latency.

Someone asked whether scheduling or hiding hardware was the worst aspect of using virtualization. Eric answered that virtualization hides the underlying hardware, but scheduling is also more difficult. He suggested that a good test to see if you are on a VM is to look for variance in the tail latency. If there is a lot of variance, you are on a VM.

Someone else asked why not run everything in a Blue Gene, and Eric said he would answer the essence of that question soon. Another person suggested just chopping off stragglers, and Eric replied that in many cases, such as MapReduce, you need all of your results.

Eric presented a strawman: that one solution to the tail latency problem would be to allocate dedicated resources for live services. That means no other jobs on those servers, so there are no scheduling issues and no page faults. The problem with this approach is that you need to dedicate resources based on your peaks, and those peaks can be 10 times as high as your average usage.

The solution has been to build huge clusters to handle the peaks, then to allocate spare capacity to batch computing. This has lead to MapReduce, Hadoop, and maybe even big data in general. Eric used Amazon pricing for spot instances as an example: on-demand instances cost 10 times as much as spot instances.

Eric then introduced Akaros, a project of the AMP Lab at UC Berkeley (http://akaros.cs.berkely.edu), a research OS made for the cloud. Akaros is a single-node OS, with scheduling decisions made by a higher layer, designed to run a mix of low-latency and batch jobs, with transparent resources. With Akaros, you provision your latency-sensitive services for peak needs, then allow resources to be allocated to all jobs' current needs. When provisioned services require resources, batch jobs lose their allocations, with a revocation time of 2–3 microseconds.

Someone asked about CPU caches. Eric replied that caches can be cold, but it is important that pages be present and not have to fault.

Akaros is designed for manycore CPUs, with as many as 100 cores per server, but it also will work for SMP. There is no user interface, so limited interrupts, careful memory partitioning, no short quanta, and all system calls are asynchronous—events are queued on syscall completion. Manycores are allocated to a single process, with a single address space, and no blocking, similar to a manycore version of Capriccio, but with context switches that are four times faster than Linux.

Eric then examined the use of VMs, beginning by saying that our need for VMs will decline over time because VMs reduce predictability and efficiency. In some cases, we will still need VMs for legacy code, server consolidation, and untrusted code. They are working on providing a VM on top of a manycore process (MCP) in Akaros.

Akaros exists now, with a 64-bit x86_64 version and a SPARC V8 version. Each application does its own local scheduling, and asynchronous system calls work. The network stack is partially done, and a Go port is in progress, as is a KVM-like VM solution.

Rik Farrow asked about the API, and Eric replied that it is POSIX. That means the Go port should be easy, but a Java port will be more difficult, as it relies on special system calls. Greg Bronevetsky (Lawrence Livermore National Lab) asked about sharing cores. Eric answered that predictable provisioning is the key. Jean-Paul else asked if user scheduling is making all the decisions, can that application get a quantum. Eric answered that the application can make a system call and set a timer, which results in an event.

Russell Williams (Adobe) wondered whether there are serious services that cannot currently be done on EC2. Eric replied that the size of many apps that are running today in large DCs are hundreds of times the size of apps running on cloud servers. Once you reach some number of servers, you will need more provisioning guarantees. Another person pointed out that cluster schedulers are already too complicated today, and that Akaros is moving into this territory. Eric replied that provisioning could make cluster scheduling simpler because you will know what resources you have.

Emery Berger pointed out that this has the flavor of hard, real-time systems, and wondered whether you will get this 100% of the time. Eric replied that that's the goal. Emery continued that what you really care about is the 99%: multiplexing, spreading out bursts across the system, performance anomalies, and embracing randomness. Eric said that they know that fan-out makes tail latencies worse. What's more likely are reactive techniques so that you can change the behavior of the system during a tail latency event. Greg Bronevetsky asked about scheduling for power usage. Eric replied that you will know in advance how much power you will be using, and that most loads are bimodal: either on or off.

Bugs

Summarized by Nuno Machado (nuno.machado@ist.utl.pt)

Characterizing Real World Bugs Causing Sequential Consistency Violations

Mohammad Majharul Islam and Abdullah Muzahid, University of Texas at San Antonio

Mohammad Majharul Islam presented a comprehensive study of 20 Sequential Consistency (SC) violation bugs, randomly collected from several well-known open source programs and libraries (e.g., Mozilla, Apache, MySQL, Java, etc.). Mohammad argued that these types of bugs are among the hardest ones to solve, mainly because, besides being counterintuitive, they result from a particular reordering of memory accesses and may only appear depending on the processor's memory model.

Mohammad reported that, by analyzing the reports of the 20 chosen bugs, they found double-checked locks to be the most common bug pattern (45%), followed by incorrect fences (30%), and improper flag synchronizations (25%). Additionally, the study has shown that these SC violation bugs can be solved by correcting fences (45%), using both atomic variables (30%) and locks (20%), and restructuring the code (5%). Mohammad also noticed the relevance of these findings by saying that around 50% of these SC bugs required from 90 days up to several years to get fixed, despite their having been caused by violations involving only two threads.

Mohammad concluded the presentation by saying that this study aims to be a substratum for further research on the topic, since the bugs collected in this work can serve as standard benchmark to evaluate future solutions intended to address SC violations.

But How Do We Really Debug Transactional Memory Programs?

Justin E. Gottschlich, Rob Knauerhase, and Gilles Pokam, Intel Labs

Justin Gottschlich centered his presentation around the question: "How does one debug a Transactional Memory (TM) program that uses real hardware?" He began by showing that traditional ad hoc debugging techniques (e.g., breakpoints, single-stepping, printf, etc.) are insufficient to cope with TMs. This is due not only to the inherent nondeterminism of multithreaded applications, but also the great complexity arising from both speculative executions and conflict management when one has software TM (STM) and hardware TM (HTM) systems.

Justin claimed that a record-and-replay (RnR) approach can be successfully employed to debug multithreaded programs, as it records the relevant nondeterministic events during runtime to, later, allow the execution to be replayed deterministically. Here, Justin highlighted their previous chunk-based RnR system, which relies on hardware support to trace the order of thread chunks (i.e., blocks of contiguous instructions executed prior to shared memory conflicts or system events); however, when it comes to HTMs, some additional issues have to be addressed. In particular, Justin pointed out the need to support tracking hardware transactional events (e.g., begin, commit, and abort); instrument STM accesses in order to record reads and writes for software transactions; and modify the replayer to provide precise information about conflicts and HTM replay emulation.

Justin finished by presenting some use cases that show how their modified RnR system can be used effectively to debug programs in which both types of transactions execute concurrently. Afterward, he was asked whether System Z-supporting hardware transactions that are guaranteed to make forward progress violates his claim that all the HTMs are used in their study. Haswell, Blue Gene/Q, and System Z are all best-effort. Justin replied that, despite System Z-supporting constrained transactions, they are limited in their size, which, in the authors' opinion, makes it too nongeneralizable a transaction type to ensure forward progress. Therefore, they still consider System Z a best-effort HTM.

Another question regarded whether the claim that all real HTMs do not have escape actions is violated because System Z supports nontransactional stores. Justin argued that no real HTM system supports full escape actions and, to the best of the their knowledge, System Z, while supporting nontransactional stores, does not support nontransactional loads; for that reason, it does not provide full escape actions.

Property-Driven Cooperative Logging for Concurrency Bugs Replication

Nuno Machado, Paolo Romano, and Luís Rodrigues, INESC-ID, Instituto Superior Técnico, Universidade Técnica de Lisboa

Nuno Machado presented a novel approach for defining instrumentation points for record-and-replay systems based on cooperative partial logging. This approach works by leveraging on multiple partial traces containing the order of access to shared program elements (SPEs), and using statistical techniques to determine which of the collected partial logs should be merged and attempted to replay the bug.

Nuno claimed that their previous work on this topic, which randomly defines subsets of SPEs to be traced, suffers from a number of shortcomings: it may not provide log overlapping, may not ensure fair load distribution, and disregards access correlations existing among SPEs. Nuno described how their new proposed technique addresses these issues. First, the technique uses a "quorum" strategy to ensure that any two SPE subsets have a non-empty intersection. Second, it performs initial training runs to measure the frequency of occurrence of each shared access, so partitions can be defined taking into account load balancing. Third, it identifies correlations between variables by computing a metric of how often they are accessed nearby within each method. Finally, Nuno mentioned that they use an Integer Linear Programming model to optimize the partition of the original set of SPEs into subsets that provide load balancing, correlations, and coverage between themselves.

Nuno finished his presentation by showing some preliminary experimental results that demonstrate initial evidence of the benefits of their new approach. In particular, for the programs tested, the approach leads to fewer attempts when trying to reproduce the buggy execution in 55% of the cases and incurs smaller load variations between partial logs.