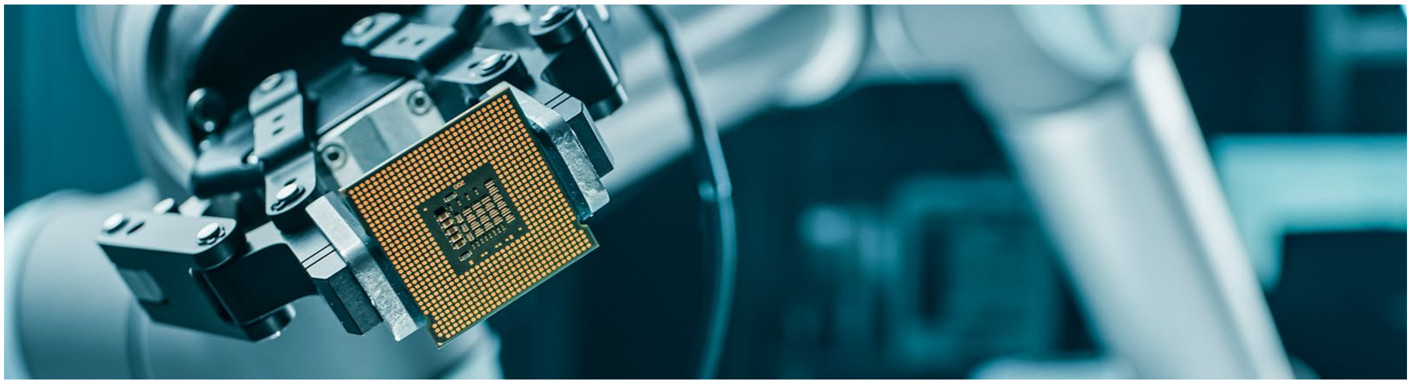
LICENSE-FIRST SCHEDULING WITH HETEROGENEOUS SCHEDULERS AND ALTAIR® ACCELERATOR™

Stuart Taylor – Altair / December 2, 2022



Get license-aware scheduling even if you're stuck with legacy or hardware-centric batch systems

Introduction

Batch systems often don't work well with complex chip design workloads. This paper covers the common problem of getting high utilization with software license resources. Established batch systems typically come from the high-performance computing space, where rich scheduling strategies are highly valued and code is often bespoke or unlicensed. It's common to treat license resources as a simple counter against which jobs can be submitted, but even in the best-run organizations this fails to deliver good utilization due to inconsistent license usage across jobs. Even a relatively sophisticated approach — consider LSF's Elim model — fails to work as things scale up. The Elim approach works when job turnover is modest compared with the linger period and polling rate of the license daemons. However, if job turnover is high — the jobs are short and/or the number of available licenses and job slots is large — then inefficiencies become noticeable. What worked for 10 concurrent jobs/licenses starts to fail as the system is scaled toward a few hundred. We might start out at 95% utilization for 10 licenses and one job start per minute, but this can drop to less than 70% if we scale to 600 licenses and a turnover of one job per second. Existing remedies are unattractive or, in some cases, dangerous. We can overcommit jobs against licenses and leverage the vendor queuing feature in daemons like FlexLM, but too many jobs in vendor queuing can result in the license daemon becoming unresponsive so no jobs get submitted.

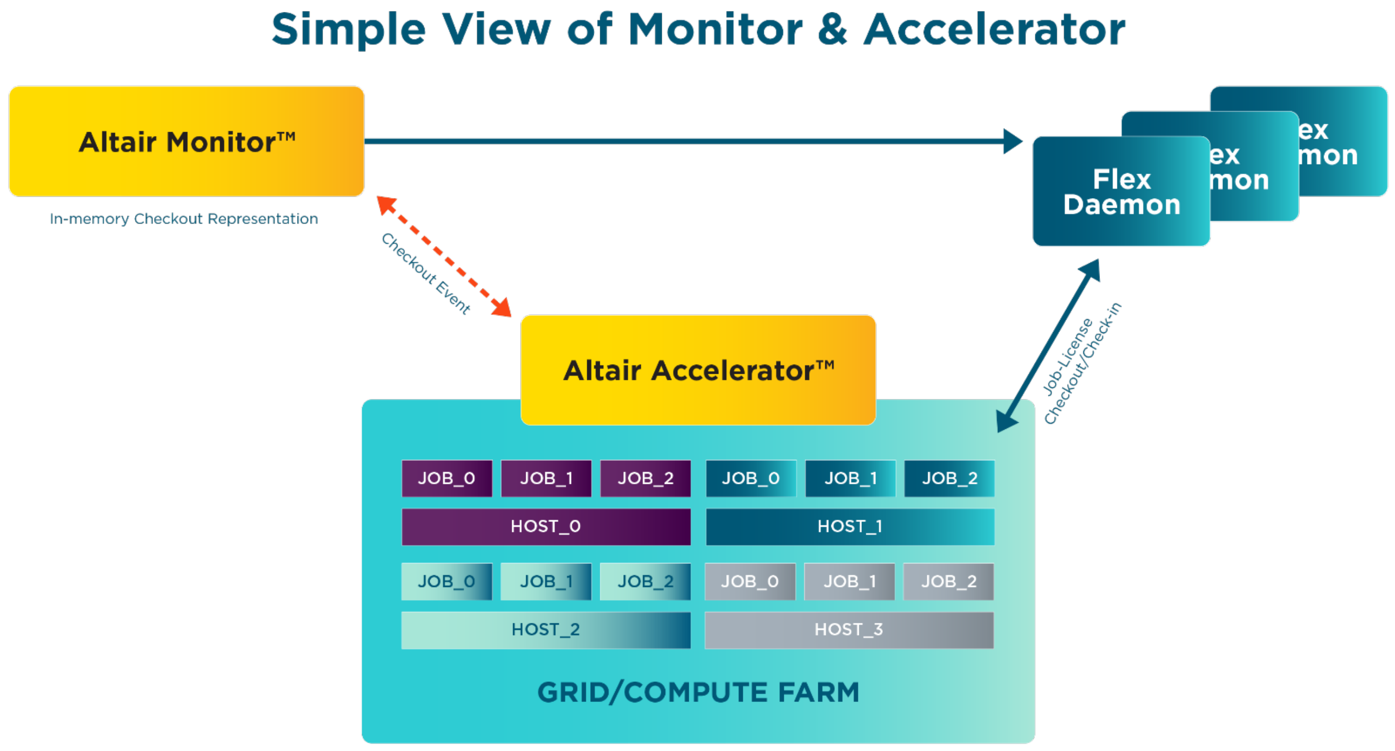
What's an alternative? Assuming we have a batch system that treats licenses as critical resources, we could use that, but it would be a disruptive change for other teams and users of the existing batch system. What we'd like to do is run the license-critical workload on a batch system that knows how to handle it, but retain the other system for other workloads. We'd also like to share the underlying hardware since partitioning is inherently wasteful.

Altair® Accelerator™ and Altair® Monitor™

Before we mix the two batch systems together let's see what we need for the license-aware scheduling workload. We need something to monitor the license daemons (FlexLM, Reprise etc.), and that's Altair's Monitor solution. While Monitor is often used to generate historical reports of license usage, it also retains an in-memory cache of current checkouts, and we'll feed that into the Accelerator batch system. Connecting the two is simple, with complexity hidden. We point the batch system's resource daemon Vov, resourced to Monitor's location (host and port), and tell it what licenses we're interested in dispatching against. Within a few seconds these become visible in Accelerator's resource tables and jobs can be dispatched against them. A typical Accelerator job submission looks like:

```
%nc run -r License:Spice#2 CORES/4 RAM/8000 – myJob.sh
```

We requested four cores, 8G of RAM, and two SPICE licenses. Software resources like licenses can be combined with hardware requests. Note that we don't need special parameters like "linger" because Accelerator's license-first scheduling takes care of that. Accelerator can also emulate LSF, so it is still possible to use bsub, at least as part of a transition plan.



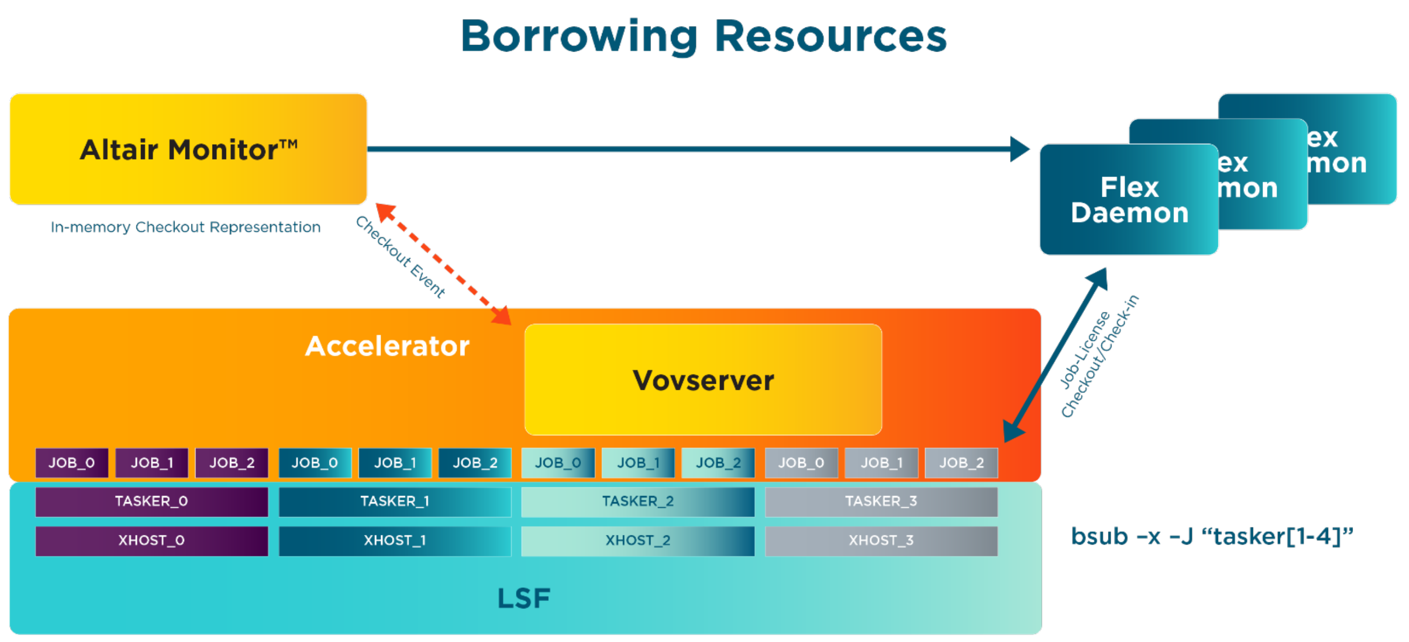
The diagram above shows the relationship between Monitor and Accelerator. Jobs run normally as within any other batch system; the interaction between the application and the license daemon is unchanged. There is no intermediation here.

Hardware Sharing

At the most basic level what we'd like to do is to borrow some hardware resources from the existing scheduler. If we are conservative, we might be concerned about interactions between the existing workload (native to LSF/Slurm) and the new workload (native to Accelerator). Using LSF as an example, to borrow we simply request resources via a bsub command. If we know that we'd like to run 500 SPICE jobs in parallel (we have 500 SPICE licenses on hand) and that we can run up to 20 jobs in parallel per host (we've benchmarked this a priori) we can determine that we'd like to borrow 500/20 = 25 hosts from LSF. We now request 25 hosts from LSF in exclusive mode, something like:

```
%bsub -q normal -x -J "taskerArray[1-25]" nc cmd vovtaskerroot
```

If the LSF queue has sufficient capacity, these 25 jobs will start running. The binary vovtaskerroot is the agent task used to start jobs on compute execution hosts for Accelerator. On startup, it will connect to the Accelerator master process (vovserver) and advertise itself as having the resources available on that host (it assumes all of them will be otherwise unused; hence the -x to bsub). At this point, assuming there are SPICE licenses available, Accelerator will start dispatching SPICE jobs. We would expect a maximum of



500 jobs to run concurrently, and a realistic dispatch rate could be in the 20-100 per second range. License utilization would be in the high 90th percentile. Nice! Even better is that LSF is not seeing this heavy load — it's only seen the 25 jobs — we're effectively reusing those bsub jobs for multiple SPICE jobs both in parallel execution (we have 20 cores/slots per job) and from job chaining. When a SPICE job finishes it is immediately replaced by another via Accelerator's job scheduler; LSF isn't involved. Once the workload is completed, the vovtasker agent process terminates and the LSF-side job completes, handing back the compute host to LSF's scheduler for other work. The borrowed resources are returned.

The above describes a perfect case. Let's consider what happens when things don't go perfectly.

Hardware Bottleneck – Here LSF can't provide all 25 exclusive hosts at once. In some cases, it might never provide all 25 concurrently even after several queued hours. What happens? Accelerator will make do with whatever it gets. Any time one of the LSF jobs starts, it will connect with Accelerator and start to receive jobs. We may not reach our target of 500 concurrent jobs, but the workload will start getting dispatched as soon as any subset of the LSF jobs start up. This is fundamentally different from running a distributed parallel job of, say, 25 hosts, for which the job will not start until all 25 hosts are available.

Request Overhang – Suppose in the hardware bottleneck case we have a couple of jobs that never get started before the simulation (Accelerator-side SPICE jobs) has completed. What happens to those jobs? Eventually they will run, perhaps after the other exclusive jobs complete, and will connect to Accelerator. When they do so, they will have no jobs to run and so enter an idle state. After a configurable amount of time, normally a few seconds, they will declare an idle timeout and exit. The borrowed resources are returned in a few seconds if they aren't used.

Out-of-Queue Usage – Let's suppose we have another LSF job that starts running and consumes a SPICE license. Perhaps it does so while our workload is executing in Accelerator. The LSF job (and especially that job's checkout) and Accelerator are essentially asynchronous at this point. We only find out that the license is not available when the SPICE job starts to run and ends up in vendor queuing. However, within a minute or so, the combination of Accelerator and Monitor will have caught up with the unexpected checkout and will mark it as Out of Queue (OOQ). At this point Accelerator will deduct this from its list of available licenses and not dispatch jobs against it. Eventually the LSF job exits, the license is returned, and within a minute or so Accelerator is back to full capacity. The approach continues to work even if large numbers of licenses become unavailable due to uncoordinated access, but we avoid excessive vendor queuing and the resulting — potentially catastrophic — load on the FlexLM daemons.

Requested Not Used – A common case, not so much for SPICE simulations but for digital implementation, is that a license isn't used all the time by a job. Accelerator can detect such cases and reconcile the license back to its free pool. This is a standard Accelerator feature, unaffected by borrowing hardware resources from another scheduler. In some cases the CAD owner of the job scripts may know when the license is no longer required. Accelerator has an API to allow a smart job to return those resources, which is more efficient than reconciling because it doesn't rely on a timeout.

Non-exclusive Hardware Sharing

The exclusive case is easy to understand and avoids risks from interactions between different workloads on the same execution host. However, it's often difficult to get hosts exclusively. We might have more success by instead of asking for 25 exclusive hosts, perhaps ask for 50 compute hosts with 10 cores or, in the most granular form, 500 one-core hosts. Accelerator supports these models, and we just need to do a little extra work in describing to the vovtaskerroot exactly what resources it can access. In general, what is requested by bsub in LSF needs to be reflected in what is advertised to Accelerator once the agent connects. If we do this correctly, both Accelerator and LSF or Slurm can share hardware resources cooperatively.

Automated Sharing

So far, we're requiring the user (or flow) to submit the workload to Accelerator and then, separately, request the resources from LSF. Can this be automated further? Yes. Accelerator and its sibling products, Accelerator Plus and Altair® FlowTracer™, have a component vovwxd that can be thought of as a coupling device between the (upper) scheduler (Accelerator in this case) and the base scheduler (LSF). This is an advanced use model and requires careful configuration to get the right policies in place, normally to satisfy the stringent requirements of LSF administrators. However, once in place it provides a fully automated way of dispatching difficult workloads that require a mix of both hardware and software resources. Note that the vovwxd is also license-aware and will only submit hardware requests if the license is available. We use the same approach to automate cloud instance bring-up, only bringing up instances if the workload is runnable and licenses are available.

Summary

Running two batch systems instead of one adds complexity. However, it may be simpler than the resulting complexity of attempting to live with one scheduler that doesn't do the job completely. The single scheduler's shortcomings are often addressed in userland with increasingly complex scripts and, perhaps worse, those scripts often impact the batch system's ability to run effectively. The idea of running multiple schedulers, if not fully fledged batch systems, is widely accepted and is often used to bring new capabilities and capacities to legacy batch systems; FlowTracer-on-LSF is a compelling example.

Accelerator-with-LSF is the idea described in this paper. While similar in many respects to FlowTracer-on-LSF, we're managing and sharing license resources within Accelerator, which allows us to aggressively dispatch jobs and drive efficiency and utilization metrics. Another key point is that this is not a stripped-down version of Accelerator. We can use advanced features such as license-to-job reconciliation and even preemption.

Realizing a More Effective EDA Environment

Semiconductor companies face daunting challenges related to chip design and verification. These challenges include increased competition; larger, more complex designs; time-to-market pressures; and limited budget for infrastructure and tools. Improvements in processor speed have slowed, causing organizations to look for new ways to improve efficiency.

Workload management is a crucial area for optimization. Even marginal improvements in workload throughput and resource utilization can drive significant productivity improvements. Altair can help organizations improve efficiency in six valuable ways:

1. **Monitor, measure, and optimize workloads.** Administrators can improve sharing policies and express resource requirements more precisely by monitoring license and resource allocations with Monitor. Improved monitoring leads to better utilization and helps ensure that critical project deadlines are met.
2. **Implement a high-throughput scheduler.** By leveraging Accelerator and its high-throughput, event-based scheduler, design teams realize reduced scheduling latency and faster job turnaround, leading to better productivity. By running jobs faster and more efficiently, they also gain simulation capacity leading to higher-quality and more thoroughly tested products.

3. **Employ license-first scheduling.** EDA tool licenses are the most valuable commodity in most verification environments, and maximizing license utilization is hard. Even sophisticated organizations may achieve only 50-70% license utilization. By using Monitor and Allocator with advanced license matching techniques, sites can dramatically improve license utilization and throughput, in some cases achieving 90% utilization or higher.
4. **Map and optimize design flows and simulations.** While multi-step workflows are common in EDA environments, many schedulers support only rudimentary job dependency management. FlowTracer captures flows, provides visualization, identifies inherent parallelism opportunities to optimize resource usage, and enables collaboration around workflow execution.
5. **Improve hardware emulation efficiency.** Most EDA firms use workload management tools, but the most expensive hardware assets in the data center — hardware emulators — are usually managed manually. Hero brings advanced scheduling and resource sharing to leading hardware emulation platforms. This provides greater flexibility and control and enables organizations to get more productivity from their hardware emulation investments.
6. **Leverage the cloud to augment on-premises resources.** While using cloud resources during busy periods sounds like a good idea, the devil is in the details. Accelerator, Monitor, and Allocator provide a complete solution for hybrid cloud deployments. Organizations can easily tap their choice of clouds to improve capacity with efficient cloud auto-scaling and policy-based software license allocation.

A Complete Suite of EDA Workload Management Solutions

Altair provides a complete suite of EDA workload management solutions to help design firms implement the six optimizations described above. Customers can start using Accelerator for high-throughput EDA workload scheduling and gradually introduce additional workload management capabilities as requirements evolve.

- Altair Accelerator high-performance job scheduler
- Altair Monitor for license monitoring and management
- Altair® Allocator™ for multi-site license allocation
- Altair Accelerator Plus high-performance hierarchical scheduler
- Altair® Hero™ high-performance scheduler for hardware emulation
- Altair FlowTracer to develop and execute design flows

Learn More

To learn more about Accelerator and other industry-leading solutions for EDA workload management, visit https://www.altair.com/accelerator/.

To learn more about Altair's Hero end-to-end hardware emulation scheduler, visit https://www.altair.com/hero/.