SYNOPSYS®
Silicon to Software™

Adaptive Resource Optimizer

For Optimal High Performance Compute Resource Utilization

July 2015

Introduction

In a typical chip design environment, designers use thousands of CPU's in a distributed high performance computing (HPC) environment to execute their tool jobs every day. As chip complexity and schedule pressures grow, ensuring compute clusters have adequate CPU and memory resources for timely and successful job execution has become mission critical. Having every machine in the HPC data center contain the fastest available CPU and maximum memory configuration may sound ideal, but it's cost prohibitive (not to mention technically challenging to maintain). To help manage design teams' requirements, IT managers have devised innovative techniques to ensure optimal usage of these HPC resources such as different job queues with memory and/or runtime restrictions. But most IT and design managers acknowledge that the compute resources used for chip design are still not efficiently used.

Surveys of HPC infrastructure managers indicate an average utilization of 50% to 60%. The highest performance machines typically have more jobs scheduled on them than they can handle, increasing the jobs pend time, which is time a job sits in queue waiting for a resource. At the same time, less powerful machines too often sit idle. This occurs because the users of these HPC clusters — chip designers — are expected to manage the process and ensure that their jobs are configured correctly to target the best machine for their job needs.

The Problem: Out of Sight, Out of Mind!

Designers must estimate the amount of resources required for their job and will often re-use flow submission scripts from other users without reconsidering resource requirements. Synopsys has analyzed nearly one million design-specific compute jobs and determined that about 96% of the jobs incorrectly estimate the memory requirement by at least 20% (see Figure 1). About 80% of the jobs overestimate the memory requirement, resulting in underutilization of HPC environment 'slots'. The number of 'slots' is the number of jobs a machine can run; a typical batch machine will have one slot per core.

On the other hand, about 12% of the jobs underestimate the required memory. Underestimating the memory can cause the HPC platform to run out of memory and start thrashing (i.e., aggressively reading and writing to and from disk instead of RAM), thereby affecting other jobs running on that cluster. In most cases, when a cluster is thrashing, jobs will run slower and hold tool licenses for a longer duration, reducing the overall efficiency of the HPC environment.

In HPC environments where queue-based runtime limits are enforced, users tend to be conservative and overestimate their job runtime to avoid their jobs from getting terminated. In doing so, queues with shorter runtime limits are underutilized and queues with longer runtime limits have a long list of pending jobs. Moreover, when flow scripts are used to submit jobs, memory and runtime requirements for all the jobs remain unchanged, resulting in underutilization of slots and inefficient use of compute resources.

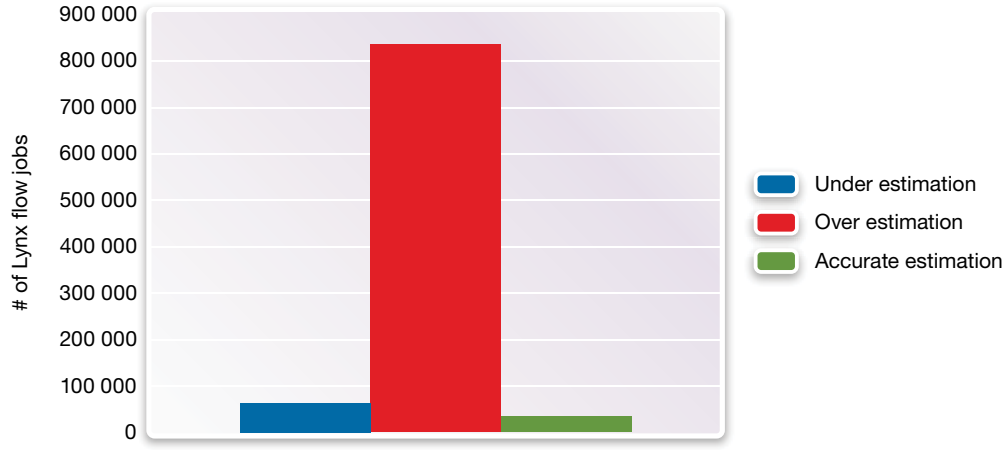

Figure 1: Ninety percent of nearly one million submitted compute server jobs incorrectly estimated the memory requirements

Adaptive Resource Optimizer: Improve HPC Efficiency

Synopsys has designed a closed-loop system that constantly assesses jobs as they are submitted and determines the correct resource needs based on past performance and automatically modifies job parameters to reflect these needs. Adaptive Resource Optimizer (ARO) predicts the required compute resources, modifies the target HPC environment if necessary, and sends the job to the scheduler.

How ARO Works

The ARO algorithm monitors usage patterns of all jobs and computes a more accurate value to be used for resource reservation of future submissions without the need for job scheduler reconfiguration. ARO can be configured to measure memory utilization or runtime or both. When the same job is submitted, ARO will dynamically modify the resource reservation before it is submitted to the HPC environment. ARO can also be configured to dynamically submit jobs to specific queues based on the computed memory and/or runtime values.

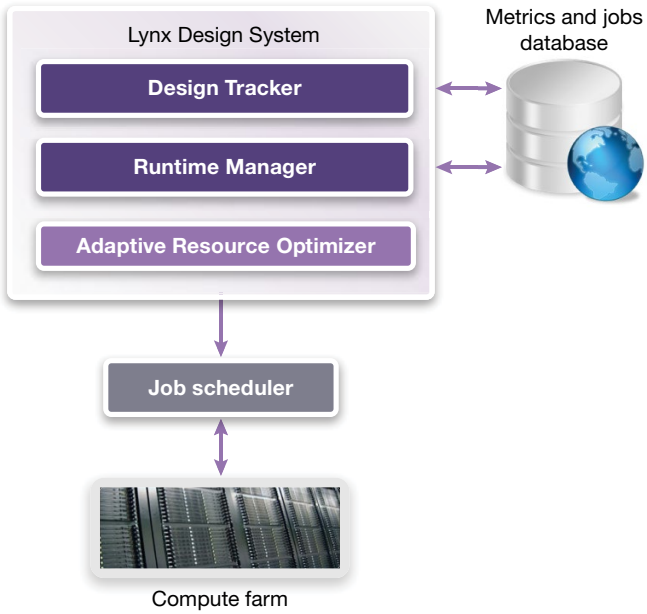

Figure 2: Adaptive Resource Optimizer: An HPC resource optimization engine

Table 1 illustrates how ARO optimizes memory usage. In this example, the job is run seven times. Each time the job is run, a new ***Job ID*** is created. ARO keeps track of the actual memory utilized for each run for the same job, shown in the ***Used memory*** column in the table. The values in ***Requested memory*** are the amount of memory requested by the designer when the job was submitted, and ***Optimized memory*** is the value that ARO determined as the optimal amount for the job. A value of -1 in the ***Optimized memory*** column indicates that optimization will be skipped for the job. For the first five runs, ARO is in "learn" mode (the number of runs in 'learn' mode is programmable). Starting at Run 6, ARO dynamically adjusts the requested memory to ensure that the job is assigned to a queue that includes machines with more memory for faster runtime.

	Job ID	Requested memory	Optimized memory	Used memory
1	1824	4000	-1	18727
2	2269	4000	-1	19117
3	2478	4000	-1	18886
4	2707	4000	-1	18886
5	3210	4000	-1	12783
6	3779	4000	22750	19118
7	4448	4000	22942	19118

Table 1: ARO dynamically increases memory requirements based on past history

Conversely, overestimating memory requirements results in underutilization of CPU resources. Some CPU cores in a multicore system may not be well utilized due to the lack of reservable memory. By dynamically adjusting the reserved memory, ARO improves cluster/queue utilization and job turnaround time—the time it takes from job submission to getting results.

Table 2 shows a report of a job that requires significantly less memory than the requested amount. As with example above, ARO is in "learn" mode for Runs 1 through 5. Starting Run 6, ARO dynamically adjusts the requested memory, lowering the request to ensure that the job is assigned to a machine queue with a smaller memory footprint and freeing up memory to be used by other cores.

	Job ID	Requested memory	Optimized memory	Used memory
1	461012	4000	-1	21
2	464851	4000	-1	21
3	469215	4000	-1	21
4	472315	4000	-1	16
5	478244	4000	-1	16
6	481397	4000	25	19
7	484722	4000	25	19
8	489021	4000	25	20
9	491831	4000	24	19
10	494594	4000	24	17

Table 2: ARO dynamically lowers memory requirements based on past history

Figure 3 shows the pending time (blue line in chart) and the turnaround time (red line in chart) for a job at different times of the day. ARO starts to dynamically adjust job parameters at the 6th run, while the runtime of the job remains the same there is a reduction of 90% in average pending time and 66% in average turnaround time. These ARO metrics can be leveraged to understand HPC resource usage and tune the number of queues and queue settings to better serve the users of this resource.

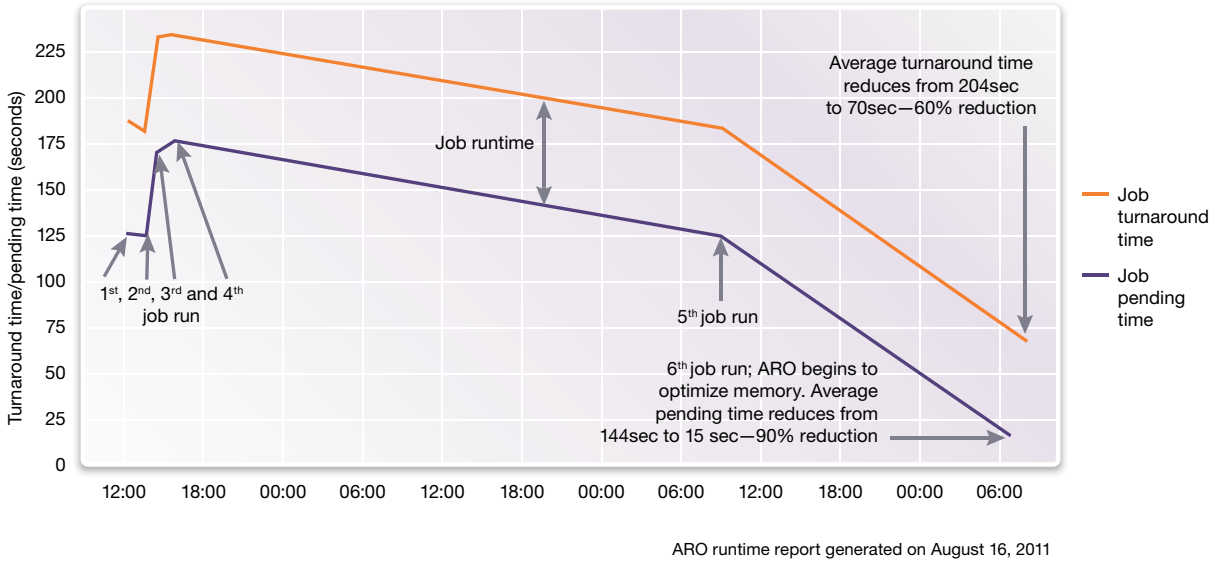

Figure 3: Trend report showing pending time and turnaround time of a job for various runs

ARO can also be configured to dynamically change the job submission queue based on an optimized memory requirement or estimated runtime of a job. ARO makes the optimal queue selection transparent to the user. This feature is called 'queue adaptivation'. Queue adaptivation is a configurable parameter and can be set for any type of HPC environment.

Queue	Run time	Memory
bigmem		>128GB
quick	< 10 minutes	
short	< 60 minutes	
normal	< 3 hours	
long	< 24 hours	
verylong	> 24 hours	

Table 3: Example job queuing strategy

In a HPC environment with a mix of queues with different runtime and memory characteristics, ARO's queue adaptivation can greatly improve overall HPC efficiency based on the various jobs' resource requirements. Table 3 shows an example job queuing strategy. In this example, ARO will place a big memory job requiring more than 128GB of memory in the ***bigmem*** queue, irrespective of the runtime requirement. Other jobs can be placed in the appropriate queue based on the runtime limit by overriding what the user provided at the time of job submission. For example, if a user decides to take a conservative approach and submit a job to the long queue, ARO will study the recent past history of the job to understand the actual runtime requirement and change the job queue to be short or normal (e.g., less than 60 minutes or 3 hours), thereby increasing the probability of getting the job scheduled on a machine faster. Typically, the number of hosts available for long jobs is much smaller than the ones available for shorter runs. This feature enables the right kind of job to be placed in the right queue, thereby minimizing any job terminations due to HPC environment policy violations and improving compute resource utilization.

ARO can be customized for each HPC cluster. Users can decide whether the job has to be processed by ARO or not by adding or removing an ARO flag at the time of job submission; similarly, a HPC environment administrator can choose to make ARO a mandatory feature and enable it for all jobs or not. ARO can be used in learning only mode to analyze the jobs that are being submitted and provide feedback to the users so that they can modify their flow and understand the job profile, without optimizing the jobs automatically.

Conclusion

ARO has been in use at Synopsys for design services projects for over two years. Since ARO has been introduced, a reduction of up to 75% in pend time of jobs submitted has been observed due to improved resource utilization. There has also been a significant reduction in the number of system reboots due to thrashing and jobs getting terminated. The busier the HPC environment, the greater the benefit realized from ARO. Improvements of 10% or more in each job's turnaround time can be achieved using ARO. When aggregated across 1000's of jobs, dramatic productivity gains can be realized across the entire design team.

ARO is available to users of Synopsys' Lynx Design System and currently validated on LSF, SGE and UGE but is adaptable to proprietary environments.

For more information, visit www.synopsys.com/lynx.