



Intelligent Multicore Resource Allocations

Improve system performance, reliability & monitoring without re-writing applications

White Paper
May 2011

Intelligent Multicore Resource Allocations

Improve system performance, reliability & monitoring without re-writing applications

Abstract

Enterprises, government agencies, research labs and universities require improved application performance to solve increasingly complex problems with greater resolution. A growing ‘multi-core dilemma’ is, however, limiting these needed application performance gains. This dilemma has been created by the architectural shift away from ever-increasing processor clock rates – a model which provided regular and ‘free’ application performance gains with each clock rate increase – to increasing numbers of cores that run at comparatively reduced clock rates. While multi-core aggregate compute capacity is greater, these many-core architectures can yield application performance levels significantly below system processing potential, as most applications are unable to benefit from aggregated compute power that spans many cores. In fact, research suggests that some applications may realize performance levels that equate to less than 15% of a system’s processing potential.¹ Therefore, to meet a given need more systems than otherwise necessary (if realized performance more closely approximated the potential) must be deployed, straining capital budgets and data center staff, space, power and cooling infrastructure. Clearly, disconnects exist between the need for improved application performance and an applications’ capability to exploit latest generation processors.

This paper addresses a solution to the multi-core dilemma via an intelligent middleware layer that dynamically learns about resource requirements and actual consumption for each task running in a system at any instance in time while simultaneously monitoring the state of system resources. This combined information, resource needs vs. resource status, is continuously fed into a queuing theory-based engine that dynamically adjusts core and memory allocations to safely yield the highest possible throughput.

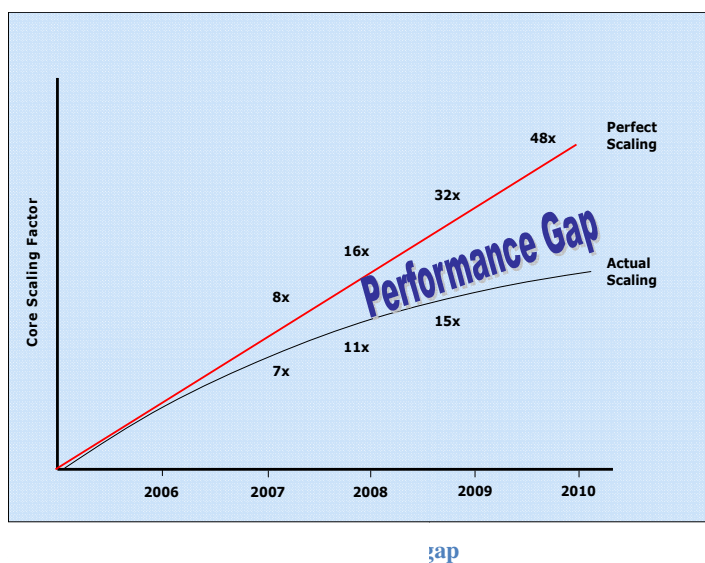
¹ IEEE Spectrum, November 2008 Edition, *Multicore is Bad News For Supercomputers*, Samuel K. Moore

Processor Shifts and the Multi-core Dilemma

Over the past 5+ years, X86 processor architectures have fundamentally changed to resolve growing power consumption and heat dissipation problems associated with processor clock rate increases. Prior generation single core processors have largely been replaced with rapidly increasing core-count architectures offering greater *aggregate* computing capacity across many processing elements. In the past twelve months, Intel (Nehalem/Nehalem-EX) and AMD (Magny-Cours) have begun to ship processors having as many as 12 cores. Today, even entry level servers, which recently averaged 4-8 cores per system, can have 24 cores or more. Published processor roadmaps for 2011/12 indicate that we will soon have 32 core processors available for deployment, which will push the average #cores per server into the 64-128 range. Importantly though, as these higher core count processors hit the market each core tends to run at slower clock rates than the single core predecessors, meaning that individual tasks take longer to complete.

Metaphorically, the computing Ferrari has been replaced by the diesel bus; individual 'passenger' speed has been de-emphasized in the multi-core era, but more 'passengers' may travel concurrently.

From an application perspective, most have been designed and written over time with the 'computing Ferrari' model in mind. As such, applications experienced metronome-like performance gains with each clock rate increase, without need for application re-writes or optimizations. But with the multi-core processor shift, these no-cost application performance gains have ended; multi-core aggregate compute capacity provides no instantaneous benefit to individual serial or lightly parallel applications. Thus, a *multi-core dilemma* has developed: most applications are unable to effectively use the capacity of latest generation processors. As Gartner Group has reported², multi-core architectures have plenty of processing power, but application design mismatches prevent users from realizing the benefits.



There is no simple solution for transforming serial designs into parallel applications, as re-writing applications is time-consuming, costly, and, importantly, not always possible, as some problems can not be solved in parallel, or may be only lightly parallel by nature. Furthermore, parallel programming is HARD, and requires different programming skill sets which very few developers currently possess.

The graph in Figure 1 illustrates the growing dilemma – see 'Performance Gap' - and application inability to harness and exploit multi-core systems. The

² InformationWeek, January 28, 2009, *Multi-Core Processors Outpacing Key Business Software*, Antone Gonsalves



benchmark depicted (VMware's VMmark) demonstrates that a single 16 core system yields about 30% less aggregate processing capacity than four quad-core systems running the same workload; with recently released 8 & 12 core processors the yield falls further. For data intensive applications the dilemma worsens, as Sandia Lab simulations indicate that 16 core systems may yield only the equivalent performance of 2 cores for this class of problem³. Given processor roadmaps with plans for 128 cores (and more) per processor, the decreasing marginal performance contribution per core must be addressed if the potential promise of multi-core processing is to be realized.

Adding to this dilemma, application throughput increases are essential to a growing class of demanding commercial applications in: finance, chip design, seismic processing, digital content creation, life sciences, etc. Industry needs a simple way to harness multi-core processing capacity in order for users to achieve a good return on processor investments, and to process these demanding applications in less time.

Multi-core processors represent the most significant information technology advancement in decades IF the aggregate processing capacity of these parallel architectures can be exploited.

Multi-core technology can dramatically: increase application processing throughput; advance server consolidation efforts; reduce system administrative burdens; lower energy consumption and related costs; and reduce data center expenditures. Realizing these attractive benefits, though, requires new solutions that can exploit multi-core system potential.

Scaling multi-core processing throughput has to this point been limited by operating system inefficiencies in allocating system resources. When one system resource (processing power) scales disproportionately in relation to other resources (bus bandwidth, memory speeds, etc.) imbalance occurs. Such imbalance creates conditions where applications can exhaust resources or conflict with one another. As the number of processing cores per system increases there is an exponentially increasing likelihood of application conflict. With only two cores in a system these conflicts are unlikely; with as few as 32 cores in a system, these conflicts are virtually guaranteed. Application conflicts lead to sub-optimal resource utilization and poor application performance.

eXludus MCOpt™ Multi-core Manager software is the first system level resource management technology developed specifically to address the multi-core dilemma, and has been designed to properly allocate system resources and prevent application conflicts. Implementing such measures enables multi-core systems to deliver their full processing capacity, and thereby significantly improves application performance and maximizes user return on investment.

A Resource Management Problem

Resource imbalance is a recurring phenomenon in computing architectures, and various methods have been used over the years to solve such problems. In the 1980's, parallelization techniques were widely used to overcome the imbalance caused by a sudden increase of processing capacity (RISC SMP) on what were essentially high-end workstations. And in the 1990's, grid workload managers

³ IEEE Spectrum, November 2008 Edition, *Multicore is Bad News For Supercomputers*, Samuel K. Moore



were introduced to overcome the emerging distributed resource allocation problem brought about by cluster, grid, and more recently, cloud deployments. However, these past methods are incapable of solving the resource allocation problem introduced by multi-core processors:

- a) Parallelization alone is neither practical (too many applications and too few parallel code developers) nor sufficient (Amdahl's Law of Serialization details the scalability limits of large core count systems). While many efforts are underway to allow for the easier parallelization of applications, *the reality is that the rate of core count increase is well ahead of the rate of parallelism.*

It is important to understand that as we deploy multi-hundred core servers in the near future, even parallelized applications will be able to use only a small fraction of the available processing power.

To illustrate, a serial application running on an 8-core server will consume 1/8 of the core resources, or 13%; a successful re-write that transforms that app into an 8-way parallel implementation (which implies very good parallelism of a particular problem) will yield an app that consumes 8/64 of a 64-core server – 13% - and only 8/128 – 6% - of a 128-core server. So, even a successful re-write will likely yield an app whose core consumption rate will actually decrease over time!

By most estimates, the majority of applications are no more than 90% parallel, which translates into an application using no more than 10 cores concurrently. So, while parallelizing applications will help, improved system-level resource allocation techniques are also absolutely necessary.

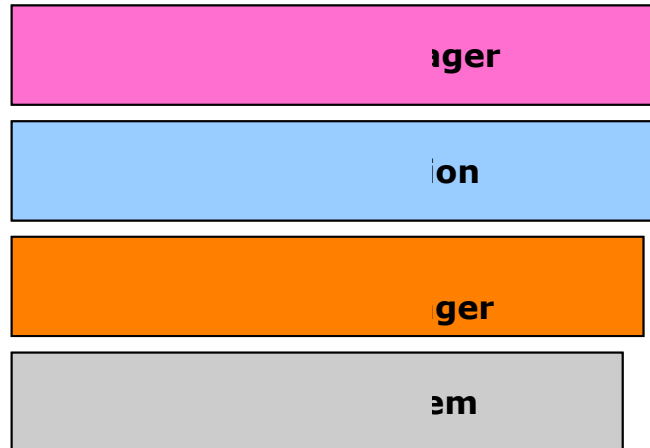
- b) Cluster/Grid/Cloud workload manager resource sampling techniques just can't keep up with the rapid pace of resource consumption changes. In addition, they fail to prevent performance limiting job conflict or interference that occurs within individual processing nodes.

eXludus' MCOpt™ is based upon operational research principles and advanced math techniques, not dissimilar from queuing theory techniques used within large scale traffic management problems. MCOpt dynamically and continuously creates an optimal processing schedule for applications such that resource utilization is maximized, and the risk of resource conflict or interference is minimized. While well-synchronized traffic lights based on relatively simplistic timings can help improve the flow of traffic through a city, MCOpt applies significantly greater intelligence to this problem. To properly extend the analogy, MCOpt is monitoring traffic coming into the city across various bridges and tunnels and monitoring the amount of traffic passing through each intersection. Then, based on the available traffic flow data and the status of the city streets, MCOpt dynamically changes the light synchronization order and timing. ***MCOpt is a dynamic and powerful resource allocation mechanism that adapts instantly to changing scenarios.***

MCOpt is not a cluster or cloud workload manager. Whereas workload managers solve the problem of distributing processing tasks across a collection of systems, they do not *and can not* control in any way how resources are allocated within these individual systems (other than by limiting the number

of jobs dispatched or possibly killing jobs that exceed certain thresholds). MCOpt, on the other hand, controls *how* system level resources are allocated and consumed by the dispatched jobs. So, cluster and cloud level (existing workload manager solutions) and system level (MCOpt) resource management solutions actually complement one another.

As a transparent middleware layer sitting between applications and the operating system, MCOpt requires no re-writes or changes to either existing applications or the operating system.

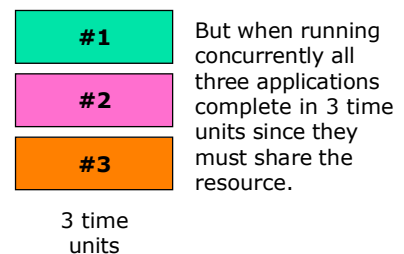
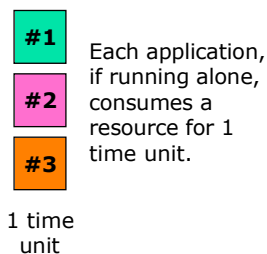


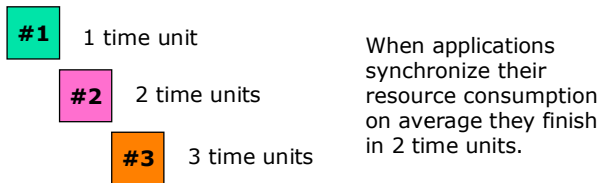
MCOpt is provided as a loadable kernel module, and does not require any system re-boot at time of install/uninstall.

Balancing Resource Allocation and Optimization

In multi-tasking operating systems, the kernel grants all resource allocation requests on the basis that processes run individually while sharing resources with other running processes. This *multi-tasking effect* has an inflationary impact on job runtimes which can become problematic in many-core servers running many concurrent tasks. For instance, given three applications that require access to a single sharable resource, each application will take almost three times as long to execute as compared to a single application running alone on the system. And, the higher the number of concurrent resource sharing tasks, the higher the average job runtimes become.

As system core counts increase, it becomes difficult to find an optimal balance that; a) maximizes system/core utilization rates, while b) minimizing the multi-tasking effect described here: too few concurrent tasks running in a system and cores will be under-utilized, and the opportunity cost of under used resources translates to longer time-to-result; too many concurrent running tasks and users experience inflated runtimes due to the multi-tasking effect and the increased contention for shared resources leading to, again, longer time-to-result. Further complicating this balancing act, if the dispatched tasks in aggregate require - even just slightly - more memory than is physically available, the operating system must start swapping pages from memory to disk, leading to significantly degraded performance (the delta between electronic memory speed and mechanical disk access speed) and even longer time-to-result!





MCOPt solves this balancing act through continuous, real-time resource scheduling algorithms that optimize system throughput based on ever changing system conditions. With MCOPt, applications collaborate – transparently - with one another when using resources in order to prevent job runtime inflation. In the Figure 4 example, applications #2 and #3

block until the first application releases the core resource, then the second application proceeds, while the third application remains blocked until the second application releases the needed resource. Average execution time is reduced (2 time units vs. 2.7 time units) through the elimination of the multi-tasking effect.

In Figure 5 the Linux multi-tasking vs. MCOPt job batching effect is easily observed, along with the benefits of the batch processing element that MCOPt injects into the system. In this particular case, a large number of *memory resident* jobs (32) were launched on an 8-core server. The blue bars details each job’s standard Linux runtime experience, and it can be seen that all jobs completed successfully in just under 150 time units. With those same jobs launched on the same system with MCOPt now enabled, it can be observed that MCOPt started to batch jobs, only allowing a job to run if there was a resource available for the job’s exclusive use. So, the first 8 jobs were assigned to a core, and these jobs completed in just under 40 time units. Then, the second batch of jobs were released from the MCOPt hold queue, and these jobs #9-16 completed in just under 80 time units. In total, it still took roughly the same time to process all jobs, **but the average job runtime under MCOPt was effectively halved**. This is a very beneficial outcome in cases where the completion times of individual jobs are important (versus the total completion time of the entire workload).

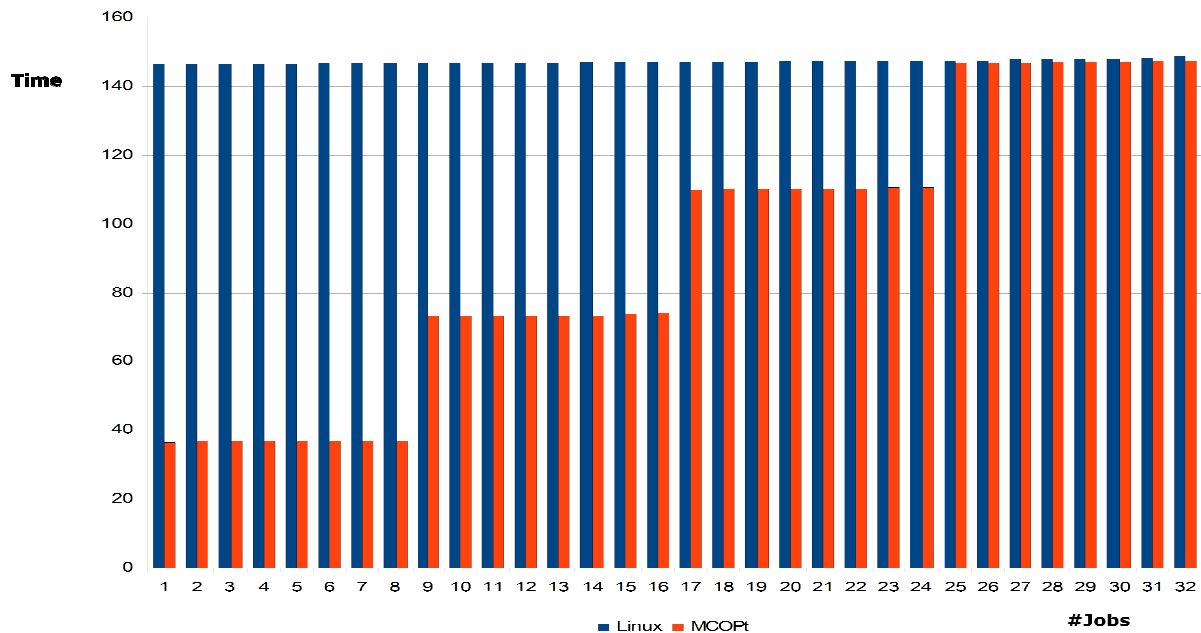
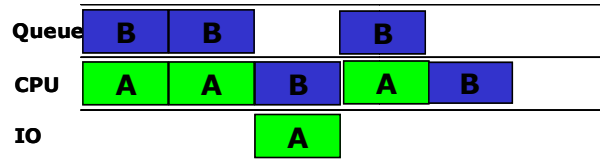


Figure 5 MCOPt job batching reduces avg runtimes

MCOPt Core ‘Scavenging’

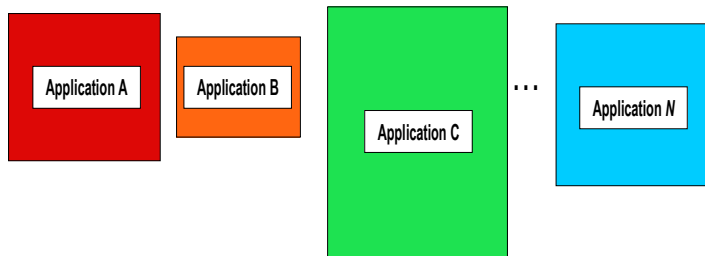
Furthermore, MCOPt recognizes that when a task *temporarily* blocks, the resources it had been consuming can be released and re-allocated, so that other jobs can benefit from that temporarily available capacity. In Figure 6 task ‘A’ is a primary, actively running job, while task ‘B’ is held in MCOPt’s job queue until the resources needed to run B become available. However, when A blocks, even temporarily, MCOPt will assign A’s core resource(s) to one or more other tasks on hold, thereby allowing those tasks to progress using resources that would have been otherwise idle. In this diagram, A executes for 2 time units and then blocks on an I/O call; B then temporarily makes use of the core while A blocks; when A unblocks it gets back its resource and B is pushed back into the MCOPt hold queue, but B has accumulated valuable runtime. Once A completes execution, B will assume the resource and begin at the prior suspension point.



Through MCOPt core scavenging, overall throughput can be increased without negatively affecting execution times, and resource utilization is always maximized.

MCOPt Core Virtualization

MCOPt manages an applications access to cores through a virtualized resource allocation system. The ultimate goal is for each application to execute as though it is running alone in a system, with no



multi-tasking effect and no/little contention with other applications vying for shared resources. In essence, MCOPt serves as a resource broker between the applications and the operating system, but implements this functionality **without requiring modifications** to the applications themselves. MCOPt continues to assign runtime environments

to new tasks until all system resources have been consumed, without over-subscribing any individual resource. Therefore, resource utilization is continuously maximized, but without risk of hitting severe performance degradation problems that can occur when a resource, such as memory, is oversubscribed. The result is *improved overall throughput* without need for application re-writes.



Note that the default MCOpt configuration is to assign 1 virtual cpu (vcpu) per physical cpu. By so doing, MCOpt will prevent core time-slicing. However, the vcpu setting is a tunable parameter that can be adjusted to site specific needs if required.

Safely Maximizing Resource Utilization

In general terms, it is not difficult to demonstrate full utilization of all system cores. Linux is a time-share based operating system, and will attempt to run as many tasks as dispatched. It's possible, then, to simply dispatch incremental work up to the point that all cores are 100% consumed.

Unfortunately, Linux has no a priori means to determine that a given scenario may lead to good, bad, or disastrous results. Even worse, the bad outcomes only tend to be discovered as (or after) they occur!

It is certainly possible for all cores to report as 100% busy while (close to) 0% is applied to application processing. (Essentially, all processing would be managing system level tasks.) So, the key to obtaining an optimal result is not in demonstrating that all cores are busy; the key is to keep cores as busy as possible applying cycles to application processing, not system management tasks - and preventing the disastrous outcomes that could otherwise occur.

MCOpt dynamically and continuously monitors system state. If MCOpt determines that a given situation will jeopardize system stability and performance, the resource manager will take proactive steps to resolve the problem.

MCOpt adds system functionality that can be viewed like that of a weather barometer; as internal system pressure rises or falls, MCOpt will automatically take proactive steps based on that condition, thereby ensuring that, given current conditions, the best possible outcome will be obtained.

One of the proactive steps that MCOpt can take is to limit the level of memory oversubscription that takes place within a system. If MCOpt detects memory pressure getting too high – a sign that performance degrading paging/swapping may start to occur – MCOpt can slow down one or more lower priority jobs, limiting the rate at which that job may build its memory footprint, or even temporarily suspend a job and force that jobs memory pages to disk, freeing memory space for higher priority work. When the pressure subsequently falls, these 'slowed' or 'temporarily halted' jobs will be allowed to continue at full rate.

To illustrate how MCOpt resolves memory oversubscription problems, 40 simulated jobs were launched on an 8-core system, and those 40 jobs in total would require about 10% more memory than is physically available. [Note: the negative performance impacts related to memory paging/swapping start to materialize with as little as 5-7% physical oversubscription.] Initially, the jobs make good progress under Linux as they start to build their memory footprint, but ultimately reach a point where physical memory is saturated and paging activity begins. The blue bars in Figure 8 demonstrate the greatly increased runtimes that occur under Linux when this paging activity begins. The red bars

representing the same jobs running under MCOpt demonstrate that the combination of MCOpt batching combined with its memory ‘throttling’ yields a much better overall result.

Oversubscribed memory leads to significant performance degradations

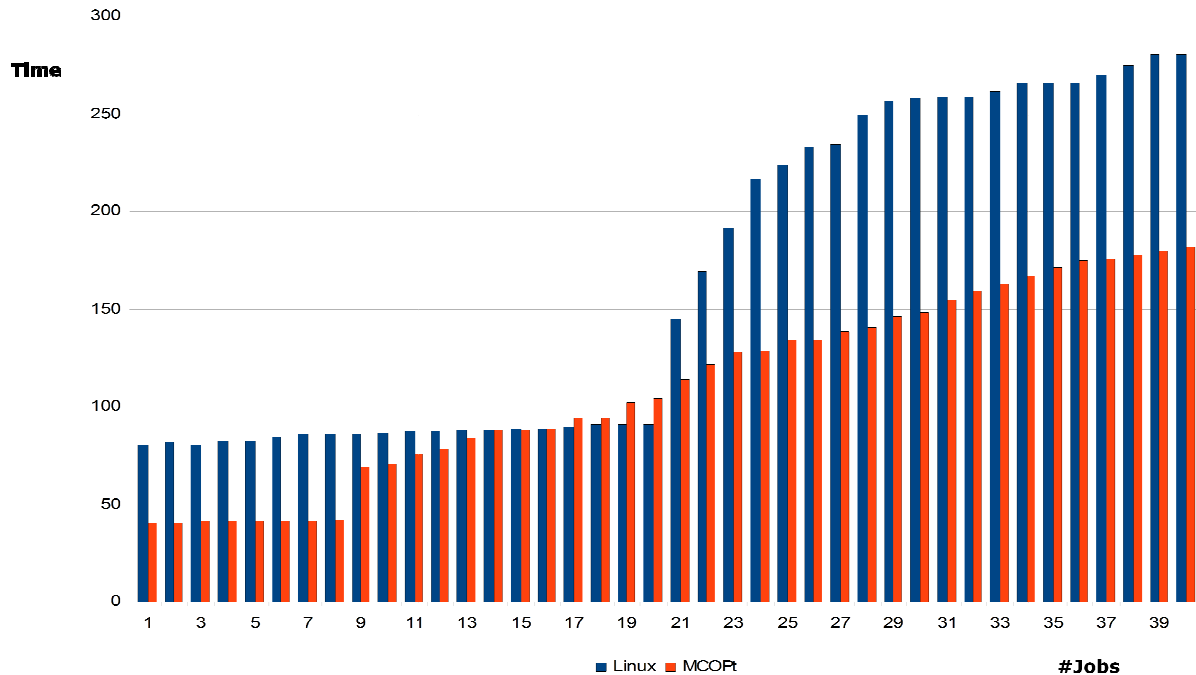


Figure 8 Performance impacts of slight memory oversubscription

Job Prioritization

MCOpt (v3.3 and higher) adds fine grain job control tools and functionality via the ability to specify various priority ranking levels for users and / or applications. Without MCOpt, the only means for freeing system resources in order to run a higher priority task is to kill lower priority jobs running in the system. While this approach does make resources available for the higher priority work, the unfortunate side effect is that the execution time accumulated by the killed processes translates to wasted cpu time (and wasted energy consumed).

Under MCOpt, if a higher priority job enters the system, one (or more) lower priority job(s) will be temporarily suspended, and the resources released by these suspended jobs will be applied to the higher priority work. When the higher priority job - or other currently executing jobs – completes, the suspended job(s) will be released and allowed to continue. ***Important jobs get the resources they need, when needed, and lost cpu time is minimized.***

[Note: In the absence of any discretely set priority, MCOpt assumes FIFO (first-in, first-out) methodology.]

An additional function that MCOpt adds to the system is the notion of an ‘exclusive job’. By setting the exclusive flag, multi-threaded/parallel jobs may accumulate multiple cores for exclusive use, eliminating scenarios where newly launched threads may begin time-slicing on cores currently in use by other unrelated tasks. This feature can be used in a manner that allows the exclusive job to immediately take access of all system cores, with any currently executing work being temporarily suspended; alternatively, this feature can allow an appropriately flagged job to gradually accumulate core resources as other work drains from the system.

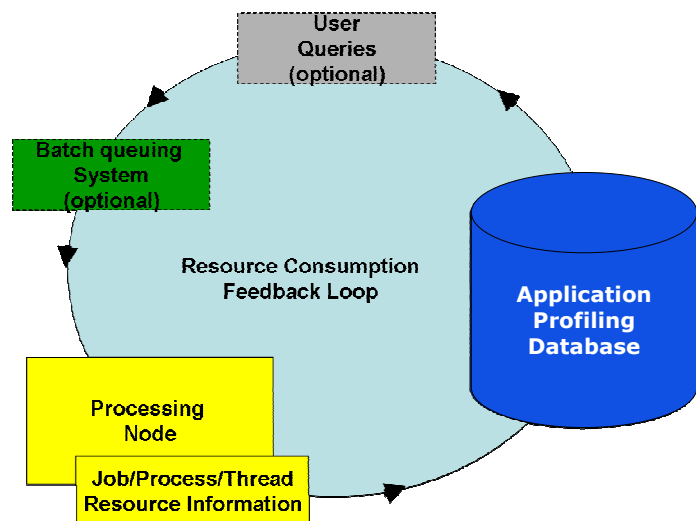
The exclusive flag can help dramatically in cases where an important multi-threaded/parallel job may otherwise sit in a workload manager queue (LSF/PBS/GridEngine, ect) for an extended time period waiting for enough jobs to drain from a system before being dispatched.

With the exclusive flag, a multi-threaded/parallel job can be dispatched to begin execution when only a single core becomes available, and the exclusive job can then immediately consume all resources or accumulate runtime on increasing numbers of cores as work drains from the system.

Information Reporting: AppProfiler™

As applications run, MCOpt learns about each job’s true resource needs by tracking resource requests and monitoring job execution. This valuable runtime information, captured per job, process, and thread, is then deposited into a central AppProfiler database (licensed separately from MCOpt) which can be queried. From this information, software developers, users and system administrators can gain insight into the actual resource needs of each task. This insight can be used to:

- Optimize software development efforts
- Schedule tasks more efficiently
- Feed automated resource hints to workload managers
- Accurately report on resources consumed, including user and system files.



Information recorded in the database can be used to generate custom queries for the purposes of:

- Service level agreement tracking
- SOX compliance
- Capacity planning

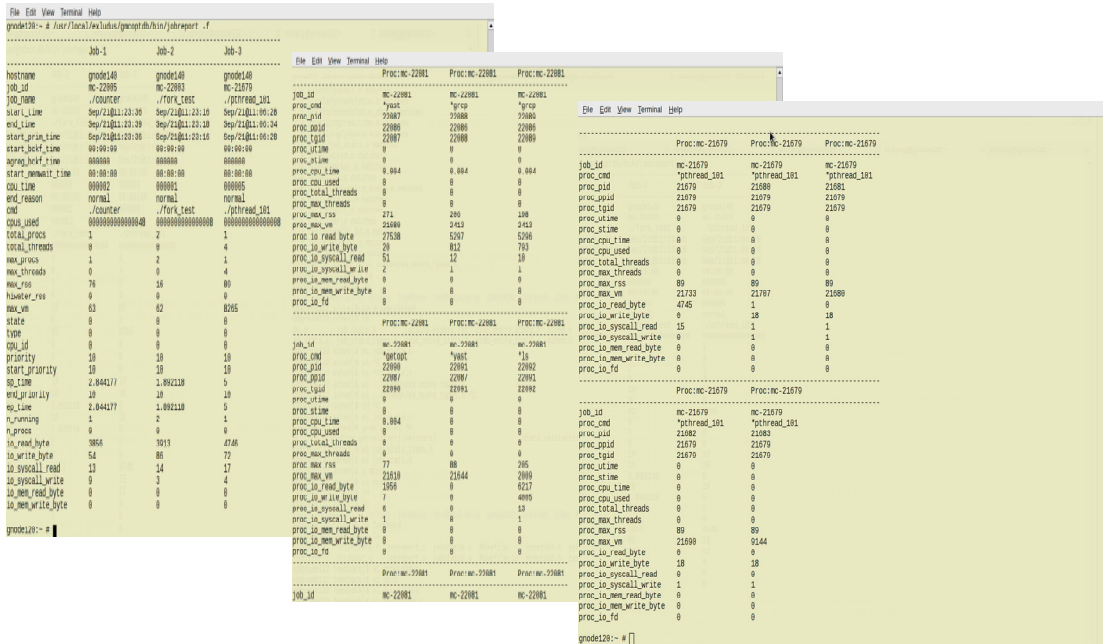


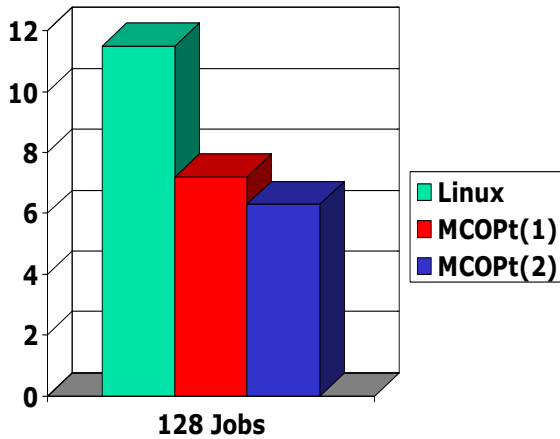
Figure 10 AppProfiler sample reports

MCOpt Scheduling Performance Benefits

Disclaimer #1: It is impossible to run any single test or benchmark that serves as a reliable proxy to any particular ‘real-world’ environment.

Disclaimer #2: It is possible to run tests that can prove any desired performance result. **In test cases where all work easily fits within the available system resources (cores and memory), and there is little I/O component to the work, MCOpt would be expected to provide an approximately neutral result versus a standard Linux deployment.** This is because there are no scheduling opportunities for MCOpt to take advantage of, nor are there any resource conflicts for MCOpt to resolve. As test cases have increasing job mixes of small/medium/large memory requirements and varying I/O needs, then MCOpt can “discover” many schedule optimization opportunities and show large performance gains. It is possible to successfully run workloads under MCOpt – especially large memory workloads – that will literally crash a standard Linux deployment.

With these disclaimers in mind, we will attempt to present a realistic representation of MCOpt value by taking a collection of applications that have varying processor, memory, and I/O requirements and creating a combined workload that includes a range of different processing tasks.



e #1

Using GridEngine (GE) as the distributed task manager, this mixed workload was dispatched to a number of dual socket, quad-core servers, and the time to complete the collection of tasks was captured, first using standard Linux and a conventional ‘one job slot per core’ setup (8 slots). [Note: most users have indicated that this “slot per core” setup is the standard used in production, which is why we have used this methodology to establish the baseline.] The baseline time was thus established at just under 12 time units (see the green bar in Figure 11). The same test case was then run with MCOpt enabled, with two different MCOpt configurations measured:

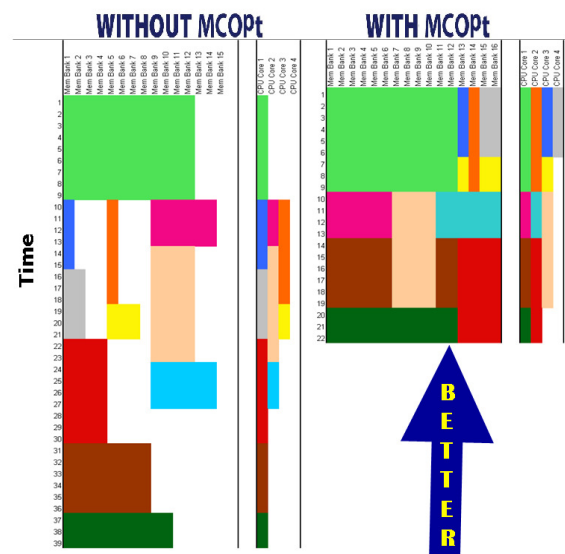
MCOpt(1) results were captured using GE to oversubscribe processing jobs to the nodes, 2X job slots per core (16 slots total). Such oversubscription would not typically be used without MCOpt, as the likelihood of performance problems related to resource oversubscription would be high. Since MCOpt only releases tasks as resources become available, this oversubscription can be done safely.

MCOpt(2) results made use of MCOpt functionality to create *two virtual processing cores* per physical core (16 virtual cores), with 2 job slots per virtual core (totaling 32 job slots). This virtual core oversubscription can be especially beneficial when a job mix contains a high proportion of jobs that have relatively low CPU duty cycles.

As seen in the results, MCOpt(1) and MCOpt(2) provided a 60% to 70% advantage, respectively, over Linux. Clearly, MCOpt made significantly better use of the available system processing power, and by putting that power to effective use the workload was processed in less time.

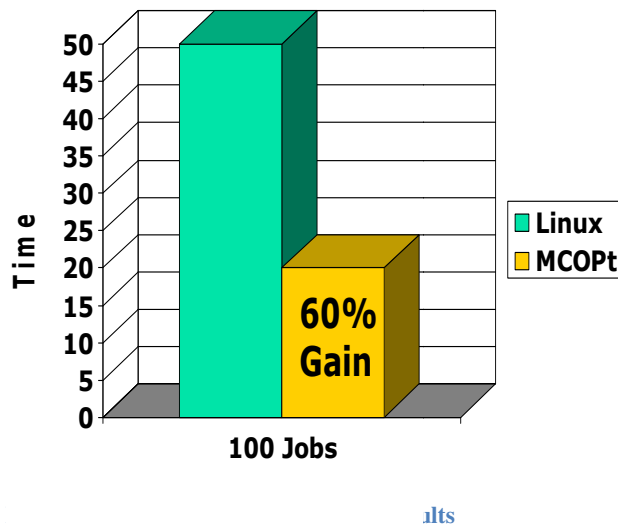
Figure 12 is another illustration of MCOpt’s potential. Here, 11 independent jobs of varying sizes execute on a cluster node dispatched first with GE alone – again using a conventional job “slot per core” configuration, *but adding workload queues for jobs designated as small, medium, and large memory jobs*, which again represent conventional techniques an environment may structure in attempt to purposefully avoid resource contention problems. The same jobs were then run with MCOpt and 2X slot oversubscription through just a single queue.

White spaces indicate unused memory or processor resources. As can be seen, with MCOpt resources are more fully used and all jobs terminate in less time, even though each job’s individual run time remains unchanged. Because MCOpt prevents the



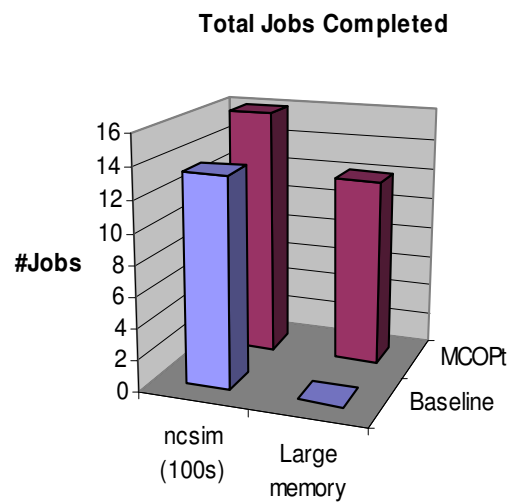
nance #2

memory swapping and associated performance degradations that would otherwise occur, the workload manager can safely push more work into the system sooner. *In additional to more fully utilizing system resources, doing so totally eliminates the dispatch latencies typically seen on a cluster.* By jobs spending less time queued in a central workload manager repository, time-to-result is significantly reduced. The net gain in this test is a near 50% reduction in total run time, or put another way, a near **doubling of system throughput!**



In tests conducted at user sites, ranging from academic grids to large scale chip design environments, **the measured MCOpt performance advantage has generally ranged from 10-70%** depending on the specific characteristics of the particular applications. One example of these tests relates to work performed at a large chip design site where a series of 100 Cadence Specman regression jobs were dispatched to dual-socket Nehalem servers using Platform LSF. By simply enabling MCOpt and increasing the number of concurrent jobs to run on each system, the time required to process these 100 tasks was reduced by 60% (Figure 13).

Another interesting observation can be derived from tests conducted by IBM that involved a mix of small and large memory jobs. In this scenario, MCOpt tends to provide meaningful gains for the small memory jobs (+17%), and significantly greater benefit to the large memory jobs (>100% benefit). Without MCOpt, the large memory jobs pay substantial swapping (moving memory pages to/from disk) penalties; because MCOpt effectively limits this swapping activity the user experiences dramatically better system performance. The results of IBM’s testing can be seen at right, where the light blue boxes represent the Linux baseline results, and the purple boxes represent the MCOpt results. The conclusion of these tests was that users could move away from strict “application silos” with work segmented to a particular silo, and move to a multi-tenant configuration which would alleviate overall cluster management and yield higher overall utilization rates.



small memory jobs



Conclusions

Although multi-core processors create resource imbalance conditions which can reduce processing efficiency, eXludus' MCOpt and AppProfiler products adjust the imbalance through intelligent resource monitoring, analysis, allocation and management. MCOpt paves the way to efficiently use the rapidly increasing number of processing cores, and the much larger multi-core processor systems anticipated.

The benefits of this technology include:

- Application throughput is improved as a direct result of making best use of the available system processing capacity and preventing/limiting application interference and memory swapping. Better application performance leads to faster results and shorter decision times.
- Consolidation efforts can be enhanced as more work can be completed within the same number of – or fewer – systems.
- Administrative overhead is reduced as users and system administrators no longer need to analyze specific application resource requirements in order to segment and isolate workloads.
- Energy efficiency is improved, as powered cores are not left un- or under- utilized, and fewer systems are needed to meet a given need.

Technology is only beneficial if it can be easily used. MCOpt is simple to deploy as installation takes only minutes, and requires neither application changes nor operating system modifications. MCOpt has no tuning parameters that users must master, nor extensive configurations to implement. Applications run “as is”. But, users get results faster!