On the Scalability of Centralized Control

Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

Abstract

Scalability of clusters and MPPs is typically discussed in terms of limits on growth: something which grows at a rate of $O(\log p)$ (where p is the number of processors) is said to be more scalable than something whose growth rate is $O(p)$. But in practice p does not grow without limits. We therefore suggest that discussions of scalability should take time into account. System sizes grow with time, so larger systems need to be supported — but only after some time. And in particular, there is no real need to support arbitrarily large systems right now. Surprisingly, when time is thus put into the picture, we find that centralized control is actually quite scalable. The reason is that the capabilities of a centralized control node grow at a fast pace due to Moore's law. This seems to be more than enough in order to manage current growth patterns displayed by parallel systems.

1. Introduction

Scalability is one of the holy grails of parallel system design. The quest for scalability and the fear of lacking scalability lie in the background of many design decisions, either explicitly or implicitly. In many cases, scalability is used as a justification for more complex designs, where a simpler design would have done had scalability not been an issue.

But what exactly is scalability? One definition is the support for incremental growth [19, 32]. With this definition, a system is scalable if you can add resources incrementally to achieve the desired level of performance. A more common definition involves the effective use of increasingly large resource pools. For example, a system's scalability could be measured by how much we should need to increase the problem size in order to maintain the same level of efficiency as on a smaller system [11]. It has also been suggested to add cost into the equation, and require that productivity scale with cost [14].

In the context of system design (as opposed to application development) a major consideration is the control structure. It is always simpler to use centralized control, whereby a centralized agent has all the required information and makes the best possible decisions. But oftentimes it is not practical to collect timely global information, and a centralized agent might become a bottleneck. For example, this is the case for the control of caching in shared memory systems. Therefore, distributed alternatives have to be designed [30].

One reason that centralized control cannot be used for caching is the required intensity: each processor must make caching decisions on each clock cycle. But in other contexts the intensity is much lower. A case in point is the management of parallel systems and clusters, including configuration management and job management. The most commonly used structure for these functions is centralized, with examples including Condor [20, 24], Prospero [23], ScoreD [12], Glunix in the Berkeley NoW [10], the ParPar cluster [7], Hector [26], STORM [9], and SLURM [33].

In a centralized control structure, system management is done by a dedicated master node (Fig. 1). Users submit jobs to the master, and the master controls the jobs execution on the different nodes. With this design the master is an obvious bottleneck and a single point of failure. There has therefore been some work on distributed and hierarchical structures, and some work on (hot swap) backups. Another alternative is to partition the load and have multiple masters with different functionalities, e.g. resource management, application support, and hardware monitoring.

In the context of scalability, the main concern is that the master node may become a bottleneck and limit the system. However, we make the observation that it takes time for clusters and parallel systems to get bigger. This is due to various technological and economic factors, e.g. the price of a single processor. Therefore scalability should not be considered with regard to a snapshot of technological and market conditions, but rather along time. During this time, the master node becomes more powerful due to Moore's law. We claim that this compensates for the additional burden on

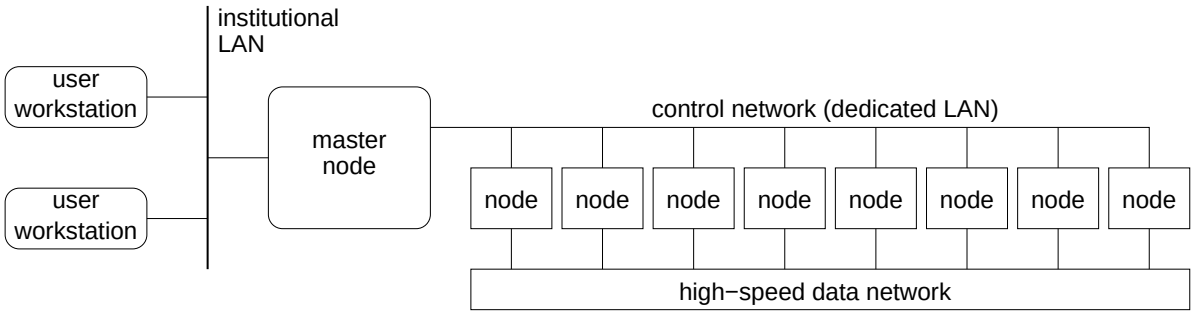

Figure 1. Schematic of typical configuration of a parallel machine or cluster.

the master, and prevents it from becoming a noticeable constraint.

In the sequel we attempt to substantiate this claim by analyzing collected system and workload data sets from the last ten years. It should be noted that this is all highly speculative, as it is hard to define and measure the things we are looking at. Nevertheless, the data seem to create a cohesive general picture, that supports the above claim.

The paper is structured as follows. The next two sections discuss configuration management. Section 2 shows that the load on the master for configuration management is proportional to the number of nodes. Section 3 then uses Top500 data to characterize the growth rate of typical installations. The next two sections after that discuss job control; Section 4 shows how the load on the master depends on the workload, and Section 5 analyzes workload data to unearth growth patterns. Section 6 concludes by comparing the growth of load with the growth in capabilities.

2. Support for Configuration Management

The typical software configuration for management comprises a single master daemon running on the master node, and an instance of a node daemon running on every other node (the names of these daemons may differ in different systems). We are interested in the load that configuration management places on the master. To evaluate this, we will use the ParPar cluster management system as an example [7, 16]. In this system, the master does the following.

Node integration. Upon startup node daemons contact the master using a well-known port. The master verifies that the nodes are in the cluster's configuration file, and executes a binding protocol with the node daemon. The protocol provides the master with up-to-date data about the nodes, e.g. their hardware capabilities. As this is done once per node, it is not a source of overhead at runtime.

Node deletion. The master keeps an open TCP link to each node daemon. If a node fails this link is severed, the master is notified, and it does some cleanup. The amount of work involved in the cleanup depends on load conditions, as it affects all the jobs that had a process on this node. However, this is a rare event and not part of normal operational overhead.

Monitoring. The simplest form of monitoring is just keeping track of what nodes are up and functioning. This can use explicit heartbeat messages among the different daemons, or rely on implicit monitoring by use of TCP. Some systems perform more comprehensive monitoring. This can include keeping track of various hardware metrics such as temperature or fan speed [17, 21]. In addition it can include scheduling-related metrics such as CPU utilization and messaging activity [31, 8].

In all the above cases, the monitoring is implemented by periodic messages between the master and the nodes. While messages from the master to the nodes can be broadcast, messages from the nodes to the master often need to be handled individually. This creates overhead that is linear in the number of nodes. This overhead is one of the reasons that a centralized master is considered a potential bottleneck, and motivates hierarchical designs [21].

Summarizing the above, we find that the overhead for configuration management is proportional to the number of nodes. We therefore need to find how the typical number of nodes grows with time.

3. The Growth of Parallel Machines

While the idea of using parallelism is very old, it has taken a long time for parallel machines to become commonly used. In terms of scalability, the interesting question is how the "typical" degree of parallelism grows with time. This is obviously a problematic question, because at any given time there are many different machines, from huge supercomputers with very many processors to desktops with only one.

One approach would be to select the biggest machine in the world, e.g. by looking at the Top500 list [1]. Note that we are not talking necessarily about the top-ranked ma-

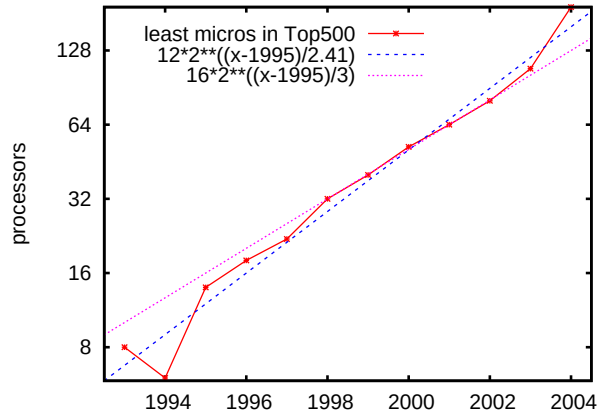

Figure 2. The size of the smallest microprocessor-based parallel machine in the Top500 list grows exponentially.

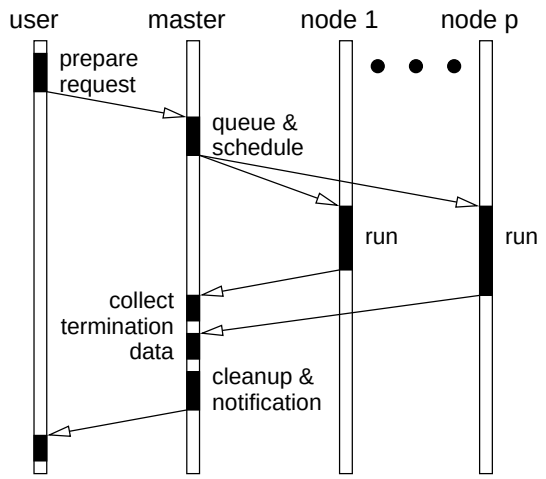

Figure 3. Sequence diagram of parallel job execution.

chine, which might be a vector machine with relatively few processors — we're interested in parallelism, so we look for the biggest. The problem is that this is obviously not representative, as the biggest machines are always unique. It may also not be timely, as a big machine can dominate the list for several years.

Due to the above considerations we suggest the alternative of using the smallest microprocessor-based machine in the Top500 list. Again, this is not the smallest machine in the list, as that is always a vector machine. It is also not the lowest ranked machine, as that could be an old machine with a relatively large number of processors. Rather, the smallest microprocessor-based machine is guaranteed to be timely and representative of the state of the art, and also to be of a size that is relatively common.

The results of tabulating this data is that the smallest microprocessor-based machine in the Top500 list grows exponentially. The data for 1998 through 2002 fits a growth rate of doubling every 3 years perfectly, as has been observed previously [4, 6]. But taking the data before and after this span into account, it seems that the growth rate is somewhat faster. A linear regression suggests a time constant of doubling every 2.41 years, with $R^2 = 0.97$.

Combined with the previous section showing that the overhead for configuration management is linear in machine size, this implies that the work performed by a management machine for configuration management grows exponentially with a time constant of 2.41 years. If at year y_0 the work for configuration management was $w_c(y_0)$, then y years later it will be about

$$w_c(y_0 + y) \approx w_c(y_0) \cdot 2^{y/2.41}$$

4. Support for Job Execution

Configuration management is just one of the tasks of the master node. The other is job management. Job execution typically has the stages shown in Fig. 3.

User request. The user activates a job representative (typically some GUI or script) and provides all the necessary details of the job to be run. This data is then sent to the master. The master performs an authentication protocol to ensure that the user can indeed run this job, and to give it the appropriate privileges. All this has constant overhead per job.

Scheduling. The master decides when and where the job will run. This typically includes the calculation of a priority function and comparison with the priorities of competing jobs. It may include queueing for some time until the required resources become available.

The overhead for scheduling may depend on the load (the number of jobs in queue that need to be considered) and on the size of the machine (the number of nodes to choose from). In simple (and common) schedulers like EASY and MAUI this dependence is typically linear in the numbers of jobs and nodes [18, 13]. In other more involved schedulers, such as the conservative, flexible, and lookahead algorithms, the dependence can be quadratic or proportional to the product of jobs and nodes [29, 22, 28]. But the constants are usually small, and in practice the overhead may be dominated by constant per-job overheads.

Startup. The master causes the job's processes to start on the designated nodes by sending instructions to the relevant node daemons. This may include copying the executable file to all the nodes [16]. The overhead involved depends on the system implementation and the job size:

- It can be constant — for example, the small-scale ParPar cluster uses an Ethernet as a broadcast medium. STORM uses efficient hardware broadcasting in a Quadrics network.
- It can be logarithmic — by use of a hand-crafted broadcast tree.
- And it can be linear — if the nodes are notified in sequence. Note, however, that the actual sending of messages to the different nodes may be overlapped.

In any case, the overhead is no more that linear in the number of nodes used by the job.

Monitoring. Monitoring the job during execution is not done in all systems. In systems that do in fact perform monitoring, it can take different forms. One example is gang scheduling, in which periodic messages are used to initiate a global context switch [12]. Another is collecting communications data to possibly change scheduling behavior [31]. The overhead is at most linear in job size, or rather in the total size of all jobs running at once, i.e. the machine size.

Termination. Unlike startup, which is one-to-many, termination is many-to-one. In most cases (e.g. ParPar) this requires separate messages from the different nodes, as they terminate at unpredictable times. The master needs to handle all these messages. An alternative is to build a reduction tree, or use hardware support for reduction as in Quadrics [9]. However, these schemes typically come at the cost of not obtaining full data about resource usage etc. If full data is collected, then message sizes grow as message numbers decrease, for a total overhead that is linear in the number of nodes.

Summarizing the above, we find that the overhead for job management is proportional to the number of nodes used by the job. We therefore need to find how the cumulative number of nodes used by all jobs grows with time.

5. The Growth of Workloads

We are interested in the total work the master needs to perform in order to support all jobs. According to the previous section support for a single job may grow linearly with job size. So we want the sum over all jobs of their sizes. For this we need to know how many jobs there are, and what their sizes are, and the relationship between these parameters and the machine size. We investigate these issues using data from the Parallel Workloads Archive (www.cs.huji.ac.il/labs/parallel/workload/). While this does not contain data from the largest classified systems in the world, we claim that the more special-purpose and larger a machine becomes, the more it is used as a capability machine, i.e. with a workload that is not very complex or dynamic (and hence easier to control centrally).

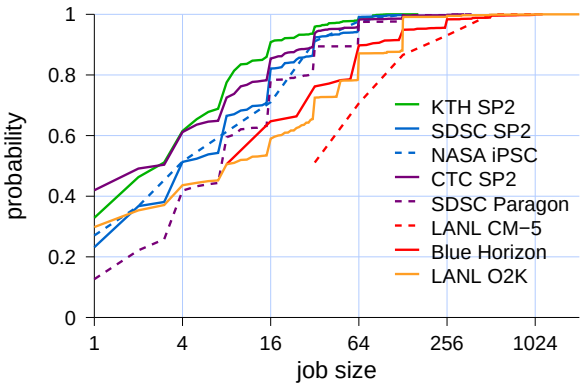

Figure 4. Distributions of job sizes.

system	*size*	*slope*	*intercept*	R^2
NASA	128	0.112408	0.276948	0.98
CTC	512	0.057333	0.586888	0.80
KTH	100	0.078803	0.518138	0.85
LANL-CM5	1024	0.096847	0.109356	0.88
SDSC-Par	400	0.103323	0.269385	0.89
SDSC-SP	128	0.105081	0.330007	0.93
BLUE	1152	0.045475	0.575054	0.76
LANL-O2K	2048	0.079070	0.308063	0.88

Table 1. Regression parameters for log-uniform model of job sizes. Slope and intercept are in $\log_2$ space.

We start with the size distribution. This is shown for several machines in Fig. 4. All have somewhere between 20% and 40% serial jobs, except for machines where the minimal size is larger than 1, which have more than 50% at the minimal size. For larger sizes the distribution is roughly log-uniform [2], but with a strong preference for powers of two, leading to a step-like shape of the CDF. The maximum size used is much smaller than the machine size, but proportional to it, so in large machines the maximum is larger.

To find the size of the largest jobs and how exactly it relates to machine size we perform a linear regression on logarithmically transformed job size data, i.e. on the distributions shown in Fig. 4, for all the machines. This is reasonable because log-uniform distributions look like a reasonable model for this data. The results are given in Table 1. From these regression models we extract the maximal job sizes — where the regression line hits 1: given a slope α and intercept β, the maximal size m satisfies $\alpha\, m + \beta = 1$, or

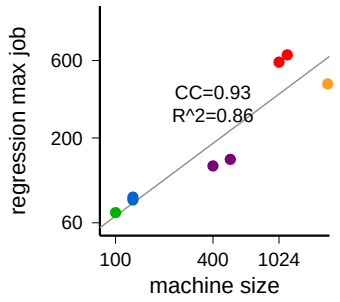

Figure 5. Maximal job size according to regression model of data from Fig. 4 as a function of machine size. Note the use of logarithmic axes.

$$m = \frac{1-\beta}{\alpha}$$

Together with the machine size p this provides a data point (p, m) for each machine for which we have data. Taking the data from all the machines leads to the scatter plot shown in Fig. 5. Performing a linear regression on these points gives slope of 0.74 in log-log space with $R^2 = 0.86$, i.e. $\log(m) = 0.74 \log(p)$. The model of how the maximal job size depends on machine size is therefore

$$m \approx p^{0.74}$$

Recall that what we actually need to estimate is the total number of processes in all jobs. If we stick to the assumption that the distribution of job sizes is indeed log-uniform, we can divide the processes in all the jobs into three groups:

1. processes in the largest jobs,
2. processes in serial jobs, and
3. processes in all other jobs (the middle jobs).

Given the log-uniform distribution there are an equal number of jobs in each binary order of magnitude, except the first (serial jobs), which has more. This can be visualized as in the left side of Fig. 6. By reorganizing as shown on the right of that figure, we find that the total processes in *all* jobs (and hence the work to support them) is approximately twice the number of processes in the largest jobs. This can be found as twice the product of the size of the largest jobs and their number.

We already know the size m of the largest jobs and how it relates to machine size. So we need to estimate the number of jobs of size m, and how it is related to machine size. This will be done by calculating how many jobs are needed to achieve a given utilization of the machine. Given this number, multiplying it by $2m$ will give an approximation of the total number of processes.

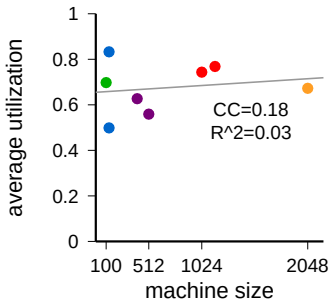

Figure 7. Utilization seems to be unrelated to machine size or period.

But what is the target utilization we should strive for? Tabulating utilization data for different machines (Fig. 7) indicates that it seems to have no relation with time or with machine size (the CTC and Paragon numbers may actually be slightly higher than indicated, as the batch partition was smaller than full machine, but this would not cause a qualitative change of these results). While some improvements in utilization have been reported in the past when schedulers were improved [15], utilization now seems rather stable at around 70% of capacity for different machines. This is probably about what is achievable with current systems for current workloads [25].

Utilization is actually the used resources as a fraction of the available resources. The resources used by a job j are the product of its size and runtime: $j.size \times j.runtime$. The overall resource usage is then the sum over all jobs:

$$load = \sum_j j.size \cdot j.runtime$$

We will approximate this by considering each job size independently, and focusing on sizes that are powers of 2 (which is reasonable based on the job size distribution data of Fig. 4). Assuming n_i jobs of size s_i, where $s_i = 2^i$ for $i = 0$ to $\log m$, denote the average runtime of jobs of this size by $\bar{t}_i$. This leads to

$$\begin{aligned} load &= \sum_{i=0}^{\log m} \sum_{j.size=s_i} j.size \cdot j.runtime \\ &= \sum_{i=0}^{\log m} n_i \, s_i \, \bar{t}_i \end{aligned}$$

To estimate $\bar{t}_i$ we return to the workload data. Previous work has identified a weak correlation of runtime and size [5, 3]. The data is shown in Fig. 8 for jobs grouped by ranges that are powers of 2, such that they have approximately equal numbers of jobs; this means that the top class includes all jobs in the tail of the distribution.

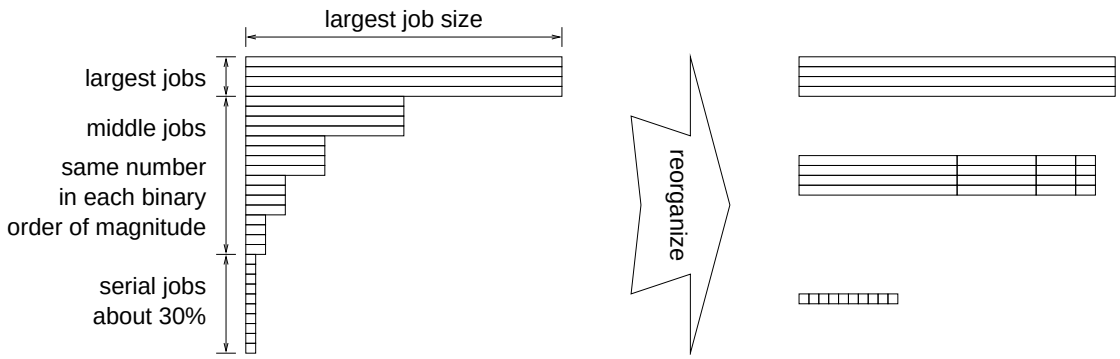

Figure 6. Estimation of number of processors in all jobs based on the log-uniform size model.

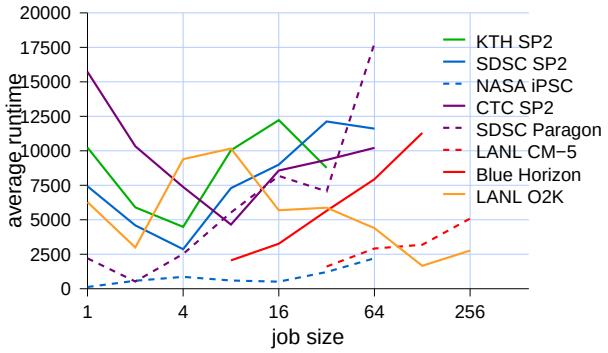

Figure 8. Average runtime as function of job size.

system	*slope*	*intercept*	R^2	*CC*
NASA	260.4	83.07	0.57	0.75
CTC	263.7	7491	0.05	0.23
KTH	1347	4241	0.47	0.68
LANL-CM5	1071	−3761	0.92	0.96
SDSC-Par	2931	−3326	0.83	0.91
SDSC-SP	1844	1458	0.86	0.93
BLUE	2316	−5538	0.98	0.99
LANL-O2K	−683.3	8442	0.29	−0.54

Table 2. Regression parameters for log-uniform model of average job runtimes, excluding serial jobs. Slope and intercept are in $\log_2$ space.

The results are somewhat noisy, but they do show an increasing trend, except for the smallest (serial and sometimes also size 2) jobs, and the LANL O2K log. Excluding serial jobs, which often have unique statistics, and computing correlation coefficients, four logs have a correlation coefficient higher than 0.9, and another two around 0.7 (Table 2). Regrettably, the magnitude of the slope varies considerably. However, for our purposes it suffices to assume some arbitrary factor α, and use the log-uniform model $\bar{t}_i = \alpha \log s_i$. But $s_i = 2^i$ and $\log s_i = i$. Plugging all this into the above derivation leads to

$$load = \alpha \sum_{i=0}^{\log m} i\, n_i\, 2^i$$

As the average runtime is correlated with job size, the time frame is set by the largest jobs. Specifically, this load (measured in node-seconds) is for the timeframe of the largest jobs, whose average runtime is proportional to $\log m$. During this time, the available resources are therefore proportional to $p \log m$. As the ratio (the utilization) is supposed to be a constant, the two expressions should be proportional to each other, so

$$\alpha \sum_{i=0}^{\log m} i\, n_i\, 2^i \approx p \log m$$

(where α is some constant, possibly different from the one defined above).

By the assumption of the log-uniform size distribution, there are equal number of jobs in each binary order of magnitude. We'll use the number of maximal-size jobs as a representative: $n_i = n_m$ for all i. As this is the same for each term in the sum, it can be taken out. We are then left with $\sum i 2^i$. Using a geometrical constructions similar to the one used above in Fig. 6, we find that

$$\sum_{i=0}^{\log m} i\, 2^i \approx 2m(\log m - 1)$$

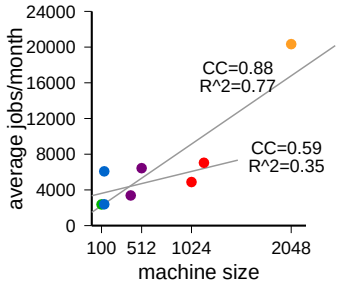

Figure 9. Larger machines tend to run more jobs per month.

Plugging this into the above expression leads to

$$2\alpha n_m m(\log m - 1) \approx p \log m$$

Discounting constants (2α), and assuming $\log m - 1$ is close enough to $\log m$ to be factored out, we are left with

$$n_m \approx \frac{p}{m} = p^{0.26}$$

indicating that the number of jobs grows slowly with machine size.

As a coarse qualitative check of this result a scatter plot of number of jobs vs. machine size is shown in Fig. 9. There indeed seems to be some positive correlation between these two parameters — larger machines do run more jobs. Calculating the correlation coefficient leads to a respectable value of 0.875. However, the LANL O2K is off the scale, and has a strong effect on this correlation coefficient. Repeating the calculation without this data point leads to a value of 0.595; somewhat lower, but still significant.

We are now ready to put all the components together and find the total work to support the processes in all jobs. Recall that this is the sum over all jobs of their sizes. This was shown to be approximately equal to twice the sum of all processes in the largest jobs, that is $2n_m m$. But we just showed that $n_m \approx p^{0.26}$, and we previously showed that $m \approx p^{0.74}$. The end result is therefore that

$$w_j \approx 2p$$

i.e. the work to support all jobs is directly proportional to the machine size. This happens because larger machines run more jobs, and these jobs are larger, but they also run longer, and in the end all these effects tend to cancel out.

Plugging the exponential growth rate of p into this result we get an approximation of how the work to support job management grows with time:

$$w_j(y_0 + y) \propto 2p_0 \cdot 2^{y/2.41}$$

As with the work for configuration management, it grows exponentially with a time constant of 2.41 years.

6. Conclusions

The previous sections made the case that the work required of the master node in a parallel system grows exponentially, doubling every 2.4 years or so. The question then is whether its capabilities grow at a commensurate rate.

According to Moore's law they should, as component density doubles approximately every 18 months [27]. However, this does not translate directly to performance. To get an estimate of performance, we turn again to the Top500 data, and look at the Rmax values of the smallest microprocessor-based machines we considered in Section 3. Performing a linear regression (in log space) indicates that this doubles every 1.09 years, with $R^2 = 0.998$. But this includes both the improvement in node performance and the increase in number of nodes. Therefore we actually have

$$\begin{aligned} Rmax(p, y_0 + y) &\approx Rmax(p, y_0) \cdot 2^{y/1.09} \\ &\approx p(y_0) \cdot 2^{y/2.41} \cdot Rmax(1, y_0) \cdot 2^{y/X} \end{aligned}$$

Factoring out the increase in number of nodes allows us to solve for X, and leads to a node capability that doubles every 1.98 years — slightly faster than the requirements. And since this is based on running the Linpack benchmark on parallel systems, we can claim that it also includes the consideration of improvements in network performance.

Thus it seems that using centralized control, in the form of a master node that is responsible for all system-wide management activities, is actually not a major bottleneck.

Acknowledgments

Thanks to Eitan Frachtenberg for his comments on an earlier draft of this paper. Many thanks are due to all those who deposited their workload logs in the Parallel Workloads Archive. It is great data to play with...

References

[1] J. J. Dongarra, H. W. Meuer, H. D. Simon, and E. Strohmaier, "*Top500 supercomputer sites*". URL http://www.top500.org/. (updated every 6 months).

[2] A. B. Downey, "*A parallel workload model and its implications for processor allocation*". *Cluster Computing* **1(1)**, pp. 133–145, 1998.

[3] A. B. Downey and D. G. Feitelson, "*The elusive goal of workload characterization*". *Performance Evaluation Rev.* **26(4)**, pp. 14–29, Mar 1999.

[4] D. G. Feitelson, "*On the interpretation of Top500 data*". *Intl. J. High Performance Comput. Appl.* **13(2)**, pp. 146–153, Summer 1999.

[5] D. G. Feitelson, "*Packing schemes for gang scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 89–110, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

[6] D. G. Feitelson, "*The supercomputer industry in light of the Top500 data*". *Comput. in Sci. & Eng.* **7(1)**, pp. 42–47, Jan/Feb 2005.

[7] D. G. Feitelson, A. Batat, G. Benhanokh, D. Er-El, Y. Etsion, A. Kavas, T. Klainer, U. Lublin, and M. A. Volovic, "*The ParPar system: a software MPP*". In *High Performance Cluster Computing, Vol. 1: Architectures and Systems*, R. Buyya (ed.), pp. 754–770, Prentice-Hall, 1999.

[8] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, "*Flexible coscheduling: mitigating load imbalance and improving utilization of heterogeneous resources*". In 17th *Intl. Parallel & Distributed Processing Symp.*, Apr 2003.

[9] E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin, and S. Coll, "*STORM: lightning-fast resource management*". In *Supercomputing*, Nov 2002.

[10] D. P. Ghormley, D. Petrou, S. H. Rodrigues, A. M. Vahdat, and T. E. Anderson, "*GLUnix: a global layer Unix for a network of workstations*". *Software — Pract. & Exp.* **28(9)**, pp. 929–961, Jul 1998.

[11] A. Y. Grama, A. Gupta, and V. Kumar, "*Isoefficiency: measuring the scalability of parallel algorithms and architectures*". *IEEE Parallel & Distributed Technology* **1(3)**, pp. 12–21, Aug 1993.

[12] A. Hori, H. Tezuka, Y. Ishikawa, N. Soda, H. Konaka, and M. Maeda, "*Implementation of gang-scheduling on workstation cluster*". In *Job Scheduling Strategies for Parallel Processing*, pp. 126–139, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

[13] D. Jackson, Q. Snell, and M. Clement, "*Core algorithms of the Maui scheduler*". In *Job Scheduling Strategies for Parallel Processing*, pp. 87–102, Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[14] P. Jogalekar and M. Woodside, "*Evaluating the scalability of distributed systems*". *IEEE Trans. Parallel & Distributed Syst.* **11(6)**, pp. 589–603, Jun 2000.

[15] J. P. Jones and B. Nitzberg, "*Scheduling for parallel supercomputing: a historical perspective of achievable utilization*". In *Job Scheduling Strategies for Parallel Processing*, pp. 1–16, Springer-Verlag, 1999. Lect. Notes Comput. Sci. vol. 1659.

[16] A. Kavas, D. Er-El, and D. G. Feitelson, "*Using multicast to pre-load jobs on the ParPar cluster*". *Parallel Comput.* **27(3)**, pp. 315–327, Feb 2001.

[17] R. Libby, "*Effective HPC hardware management and failure prediction strategy using IPMI*". In *Proc. Linux Symp.*, pp. 291–300, Jul 2003.

[18] D. Lifka, "*The ANL/IBM SP scheduling system*". In *Job Scheduling Strategies for Parallel Processing*, pp. 295–303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.

[19] G. J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons*. John Wiley & Sons, 1987.

[20] M. J. Litzkow, M. Livny, and M. W. Mutka, "*Condor - a hunter of idle workstations*". In 8th *Intl. Conf. Distributed Comput. Syst.*, pp. 104–111, Jun 1988.

[21] M. L. Massie, B. N. Chun, and D. E. Culler, "*The ganglia distributed monitoring system: design, implementation, and experience*". *Parallel Comput.* **30(7)**, pp. 817–840, Jul 2004.

[22] A. W. Mu'alem and D. G. Feitelson, "*Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*". *IEEE Trans. Parallel & Distributed Syst.* **12(6)**, pp. 529–543, Jun 2001.

[23] B. C. Neuman and S. Rao, "*The prospero resource manager: a scalable framework for processor allocation in distributed systems*". *Concurrency — Pract. & Exp.* **6(4)**, pp. 339–355, Jun 1994.

[24] J. Pruyne and M. Livny, "*Interfacing Condor and PVM to harness the cycles of workstation clusters*". *Future Generation Comput. Syst.* **12(1)**, pp. 67–85, May 1996.

[25] L. Rudolph and P. Smith, "*Valuation of ultra-scale computing systems*". In *Job Scheduling Strategies for Parallel Processing*, pp. 39–55, Springer Verlag, 2000. Lect. Notes Comput. Sci. vol. 1911.

[26] S. H. Russ, J. Robinson, B. K. Flachs, and B. Heckel, "*The Hector distributed run-time environment*". *IEEE Trans. Parallel & Distributed Syst.* **9(11)**, pp. 1102–1114, Nov 1998.

[27] R. R. Schaller, "*Moore's Law: past, present, and future*". *IEEE Spectrum* **34(6)**, pp. 52–59, Jun 1997.

[28] E. Shmueli and D. G. Feitelson, "*Backfilling with lookahead to optimize the performance of parallel job scheduling*". In *Job Scheduling Strategies for Parallel Processing*, pp. 228–251, Springer-Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.

[29] D. Talby and D. G. Feitelson, "*Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling*". In 13th *Intl. Parallel Processing Symp.*, pp. 513–517, Apr 1999.

[30] S. Thakkar et al., "*New directions in scalable shared-memory multiprocessor architectures*". *Computer* **23(6)**, pp. 71–83, Jun 1990.

[31] Y. Wiseman and D. G. Feitelson, "*Paired gang scheduling*". *IEEE Trans. Parallel & Distributed Syst.* **14(6)**, pp. 581–592, Jun 2003.

[32] M. Yang and L. M. Ni, "*Incremental design of scalable interconnection networks using basic building blocks*". *IEEE Trans. Parallel & Distributed Syst.* **11(11)**, pp. 1126–1140, Nov 2000.

[33] A. B. Yoo, M. A. Jette, and M. Grondona, "*SLURM: simple Linux utility for resource management*". In *Job Scheduling Strategies for Parallel Processing*, pp. 44–60, Springer Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.