M.Tech Thesis - First Stage Report

Tuning of High Performance Computing Systems

Ramya Raghavendra
Advisor: D. Manjunath

July 2, 2008

Abstract

Recent developments in e-commerce and web-based applications have inspired 'performance tuning' of computing systems. Typically, the workload offered to a computer system has different requirements. The rate of change of the workload is also very rapid. Thus, there is a need to track the changes and allocate resources autonomically. Tuning techniques like the simplex and the stochastic optimization techniques have been considered and their convergence properties have been investigated on a queuing model of a computer system. The stochastic optimization technique performs better when compared to the simplex techniques.

1 Introduction

Recent advancements in web-based application and services have increasingly impressed upon the need for autonomic computing or self-managing systems. Self-managing systems are those that are capable of setting their parameters of interest without any human intervention. Some examples of the parameters of interest are the total number of threads allocated to a particular process and the total number of jobs in the system.

The workload offered to an application can be classified into different categories depending on their resource requirements. The self-tuning mechanism should be able to track the change in the workload, preferably automatically, especially when the rate of change is rapid. This motivates the need for self-tuning of computing systems as the server needs to allocate its resources on a need basis which in turn depends on the class of request coming in.

The project explores different optimization techniques for tuning a computer system. We have used queuing models to model a computer system. A basic model describing the computer system has been considered for tuning. Our tuning algorithm is basically an admission control mechanism in which we restrict the number of customers in the system. The following sections describe the related literature and our attempt to tune a model computer system.

The report is organized along the following lines:

- Workload Classification
- Performance Measures
- Optimization Algorithms for Tuning
- Queuing Models

2 Workload Classification

The resource requirement of an incoming request depends on the type of workload it belongs to. Thus, the type of workload is crucial for the tuning process [3],. A possible classification of workloads is to determine whether the incoming workload is an OLTP (Online Transaction Processing) or a DSS (Decision Support System) workload. An OLTP workload is characterized by transaction oriented applications in which a number of data entry and retrieval operations

are involved. This type of workload stresses the Input/Output (I/O) subsystem of a computing system due to its frequent I/O accesses. A DSS workload is characterized by operations which are larger and longer and more CPU intensive. Thus a DSS workload stresses the CPU of a computing system. The typical transactions in a month for a banking system is an example of an OLTP kind of workload; while preparing financial reports at the end of the month is a DSS-like workload [3]. Other examples of OLTP and DSS include the TPC-C and the TPC-H benchmarks and the 'browsing' and the 'ordering' profiles from the TPC-W benchmark.

In practical systems, no workload is purely OLTP-like or DSS-like. Typically, a mix of the two workloads is presented to a server. The workload classifier, however, needs to find out the percentage of DSSness or OLTPness in the workload [3].

In our model we consider a system in which we have two classes of workloads with one class stressing the I/O subsystem and the other class bottle-necking the CPU subsystem.

The computing system services the requests (workload) that are input to it. While doing so, we need to ensure that the system is behaving in an acceptable manner, for instance, a desired level of throughput may need to be maintained or the response time needs to be lower than a given threshold. In order to do so, we need to define some performance indicators which tell us how well the system is handling its workload.

3 Performance Measures

Typically, throughput, response time and blocking probability are taken to be good characteristics describing the performance of a system [7].

Throughput measures the number of packets receiving service from the system. Response time is the total time spent by a request, which is a sum of its time spent waiting for service and receiving service. Blocking probability is the probability that a request is not taken into the system.

Other measures of performance include the 'Fairness Index' [5] and 'Power of a Queue' [7].

3.1 Fairness Index

This is a quantitative measure of how fairly the system is treating its users. This index lies between 0 and 1. An index value of 1 indicates that the system is 100% fair to all its users and a value of 0 states that its is completely unfair. 0.2 indicates that the system is unfair to 80% of its users. This index is independent of the type of resource that is being shared. The expression for the Jain's fairness index is given by

$$JFI = \frac{[\sum_i x_i]^2}{\sum_i x_i^2} \tag{1}$$

where x_i is the share of the resource received by the user i in the system.

However, using fairness index may not be acceptable in some situations as value of the index is close to 1 even when the resource distribution is 'equally unfair' among all the contenders.

3.2 Power of a Queue

This performance index combines two of the most commonly used measures for performance. The Power of a queue is defined as the ratio of the throughput to the normalized response time for a lossless system.

$$P = \frac{\lambda}{R} \tag{2}$$

where, λ gives the throughput of the system and R gives the response time of the system.

As the input load to a system increases, the throughput increases, but the response time increases as well. There needs to be a trade-off between the two. The power function is brings about this compromise. We use a modified power function in our objective function. This modification was done in order to have better utilization of the resources at the optimum. The generalized power function is as follows

$$P = \frac{\lambda^\alpha}{R} \tag{3}$$

where α is any number that can be set according to one's parameter of interest. The maximum of the power function shifts to the right with the increase in the value of α.

We have used the power function and the fairness index to describe the objective function that we would like to maximize. Our objective function for a two-class workload is as given below.

$$F(k1, k2) = a_1 P_1 + a_2 P_2 + a_3 JFI(P_1, P_2) \quad (4)$$

where, P_1 is the normalized power for customers of class one and P_2 is the normalized power for class two customers. $k1$ and $k2$ are the number of customers of class one and class two respectively. a_i are the weights associated with each term in order to make the objective function convex with respect to $k = (k1, k2)$. a_i also describes which quantity in the expression is of greater importance to us. For instance, if P_1 is to be maximized, we could use a greater value for a_1 than for a_2.

The following section describes the optimization techniques that we have used to find the maximum of $F(k1, k2)$.

4 Optimization Algorithms for Tuning

The Optimization algorithms are techniques that are used to find the value of the vector $U = (u_1, u_2...u_i)$ such that a function $f(U)$ is either maximized or minimized. They work when a convex relationship exists between the objective function and the variables involved.

Gradient based optimization functions use the gradient as the direction of maximum rate of change of the function and proceed in that direction to find the optimum of a function. But, the computation of the gradient becomes a very tedious task when the objective function is multi-dimensional. Hence we look at 'direct search' techniques which do not involve computation of the gradient. These techniques require very little knowledge of the target system that needs to be tuned. Hence, these techniques could also be termed as model-free optimization techniques.

Some techniques which fall into this category are the simplex techniques and the stochastic optimization techniques.

4.1 Simplex Techniques

A simplex is a geometric figure in the space formed by the configuration parameters. A point in that space is defined by U given by $(u_1.u_2, ...u_n)$ where u_i gives the value of the parameter i. The value of the objective function is given by $f(U)$. We discuss two kinds of simplex techniques, namely

- Nelder-Mead Simplex Technique
- Anderson and Ferris Direct Search Technique

The Nelder-Mead simplex procedure for function minimization [8] evaluates a simplex in each iteration. The value of the objective function is evaluated at each point in the simplex. The point where the value of the function is maximum is removed.

Consider a two dimensional simplex with vertices A, B and C. Let us assume that the simplex is sorted such that $f(C) \leq f(B) \leq f(A)$, where $f(A)$ is the objective function at point A. $f(E)$ is compared with $f(C)$ and $f(B)$ and $f(A)$. An extended point is considered in the direction AE if E is the best point. If E is an intermediate point, the point A is replaced by point E. If E is worse than point A, the point to be replaced, the simplex is contracted to form BCG. The figure explaining the algorithm is given below.

The Anderson-Ferris direct search algorithm is an optimization technique used when the function evaluation is inaccurate [1]. Unlike the NM-simplex method, this method pivots the simplex or the structure about the point where the cost function is at its least value (considering objective function minimization). The algorithm initially reflects about the best point and expands in case the new structure has a cost function thats lower than the original cost function. The structure contracts in case the cost function is greater than that of the original simplex. A variable called the *level* ensures that the size of the structure lies within reasonable limits.

Of the two simplex techniques, we have found that the Nelder-Mead algorithm gives better convergence. The reason as to why this behaviour is observed is yet to be investigated.

4.2 Stochastic Optimization

Stochastic Optimization techniques deal with the iterative algorithms that are used to find the roots of an equation or to find the maximum of an objective function under a noisy environment

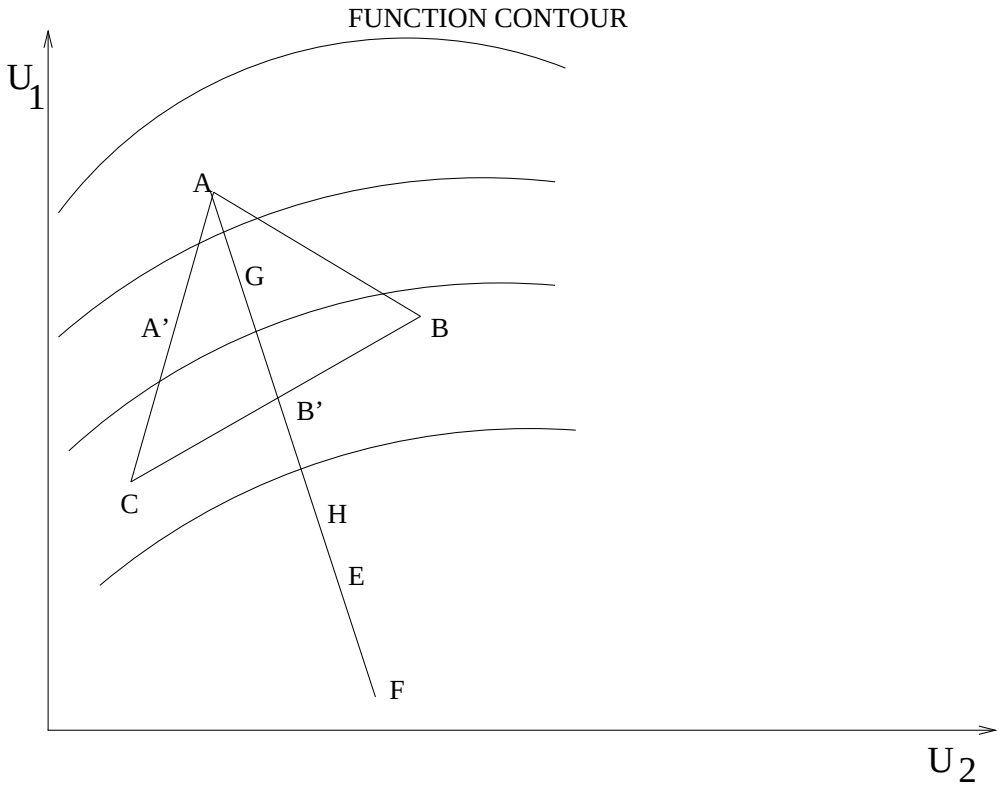

Figure 1: Nelder-Mead Simplex Technique for Function Minimization

[11], [6]. The Keifer-Wolfowitz algorithm iteratively finds the value $z = \theta$ such that the function $y(z)$ is maximized. The procedure generates a set of values z_n such that it converges to θ in probability.

The update scheme is given by

$$z_{n+1} = z_n + a_n \frac{y(x_n + c_n) - y(x_n - c_n)}{c_n} \tag{5}$$

where a_n can be treated as a step size parameter. a_n and c_n are such that

- $\sum a_n = \infty$
- $\sum a_n c_n \leq \infty$
- $\sum (\frac{a_n}{c_n})^2 \leq \infty$

We have observed that the Keifer-Wolfowitz algorithm performs much better when compared to the simplex techniques in a noisy environment. However, the performance of the algorithm depends heavily on the choice of the step size parameter. A small value for the parameter is desirable always to ensure better convergence.

We now proceed to describe the target system which was tuned using the algorithms described above.

Several techniques have been used for performance evaluation like benchmarking, simulation methods and analytical queuing models. Benchmarking is the most expensive and the most complicated out of the approaches. However, this technique does not provide us with a rich mix of the different classes of workloads and is not a very suitable platform to test the tuning algorithms. Hence, we have used analytical queuing models to describe our target computer system.

The following section provides a brief overview of the different types of queuing models and explains the queuing model that we have considered for tuning.

5 Queuing Models

A single server queuing model can model a system in which a customer demands a certain amount of service and leaves the system after receiving it. In more complex situations like a computing system, where resources are being shared among different contenders we need a

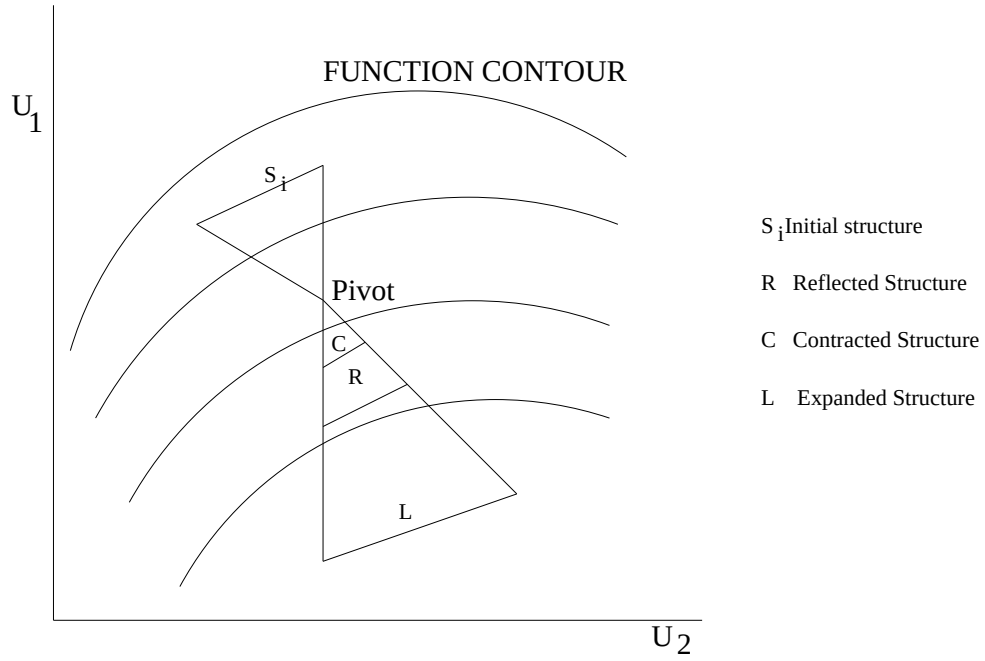

Figure 2: Anderson-Ferris Direct Search

network of single or multiple server queues [4]. A server or station (also called a node) represents the resource like CPU or the I/O channel. Customers move from one node to another as they receive service. Systems in which the customers eventually leave are called 'Open Queuing Systems' and the ones in which there are no external arrivals are called a 'Closed Queuing System'.

Queuing networks with multiple nodes and multiple input classes are solved using BCMP (Baskett, Chandy, Muntz, Palacios) theorem [2].

5.1 BCMP Theorem

Consider a network with N nodes and R job classes. The jobs may change class as they get serviced from one node and move to another. Let $p_{ir,js}$ denote the probability that a job of class r at node i moves to node j as a class s customer. The probability that the job leaves the network is $p_{ir,0}$ given by $1 - \sum p_{ir,js}$ Let i represent the node number and r represent the class of the request. The state of the system N is given by $N = (n_1, n_2, ...n_i)$ where n_i is the number of customers at node i.

The general solution [2]for the steady state distribution of the system is given by

$$f(\mathbf{S}) = \frac{1}{G} d(\mathbf{S}) f_1(\mathbf{S_1}) f_2(\mathbf{S_2}).....f_n(\mathbf{S_n}) \tag{6}$$

The factor $f_i(\mathbf{S_i})$ depends on the type of scheduling discipline thats followed at the node. G is the normalizing constant. $\mathbf{S_i}$ denotes the state of the node i.

One the basis of the scheduling strategy adopted, the nodes can be characterized as

- First Come First Serve (FCFS)
- Processor sharing (PS)
- Server Per Job (SPJ).

For an FCFS station, the node state is defined as $(r_1, r_2, ...r_n)$ where n is the number of jobs present and r_i represents the class of the job with index i.

For a PS node and a SPJ node, the state is defined by $(a_1, a_2...a_R)$ where a_r is defined as $(n_{ir1}, n_{ir2}...n_{irl})$. n_{irl} denotes the number of class r jobs at node i in the l-th stage of service. R is the number of classes present in the system.

f_i for FCFS queues is given by

$$f_i(\mathbf{S_i}) = \prod_{r=1}^{n} \left[\frac{e_{ir}}{\mu_{ir}} \right] \tag{7}$$

f_i for PS and SPJ nodes is given by

$$f_i(\mathbf{S_i}) = (n_i)! \prod_{r=1}^{R} \frac{(e_{ir}/\mu_{ir})^{n_{ir}}}{n_{ir}!} \tag{8}$$

where the stage of service l is not considered.

$d(\mathbf{S})$ is 1 for a closed system.

The normalizing constant G is given by

$$G = \sum_{\forall n_{ir}\ s.t.\ \sum_{i=1}^{i=n} n_{ir}=K_r} (f_1(n_{11}, n_{12}....n_{1R}) f_2(n_{21}, n_{22}...n_{2R}).....f_n(n_{n1}, n_{n2}..n_{nR})) \tag{9}$$

where K_r is the number of class r jobs in the network.

Calculating the normalization constant in this way is very tedious and compute-intensive. So, we use the discrete convolution method as suggested by [10]. The method is as follows

$$\begin{aligned} G' &= P_1 * P_2 * P_3.....P_n \\ G(K_1, K_2...K_r) &= G'(K_1, K_2...K_r) \end{aligned} \tag{10}$$

where,P_i is $[f_i(n_{i1}, n_{i2}...n_{iR})]$, $ij = 0, ..., K_j$

The different measures of performance, namely the throughput and the response time can be evaluated as follows using Little's law.

The throughput denoted by λ_r for a class r customer.

$$\lambda_r(\mathbf{n}) = \frac{n_r}{Z_r + \sum R_{r,i}(\mathbf{n})} \tag{11}$$

where, $\mathbf{n}$ is the vector $(n_1, n_2...n_n)$ where n_r denotes the number of customers of class r. Z_r denotes the think time of the customers. $R_{r,i}$ denotes the response time of the class r customers at node i.

The queue length of a given class of customers is given by,

$$Q_{r,i} = X_r(\mathbf{n}) R_{r,i}(\mathbf{n}) \tag{12}$$

The response time of the customers of class r is given by

$$R_{r,i}(\mathbf{n}) = D_{r,i}(1 + A_{r,i}(\mathbf{n})) \tag{13}$$

where,$A_{r,i}(\mathbf{n})$ is the queue length as seen by a class r customer into the node i.

5.2 Our Queuing Model

We have considered a closed queuing network for our target system with two classes of workload. Class 1 stresses the CPU and class 2 stresses the I/O subsystem with the total number of jobs being K_1 and K_2 respectively. The CPU requirement and the probability of disk access of the two classes is different. Node 1 represents the inactive threads in the system and is an $M/M/\infty$ queue. Node 2 represents the CPU and is a $M/M/1$ queue with egalitarian processor sharing scheduling procedure. Node 3 represents the I/O and is a $M/M/1$ FIFO queue. The Figure 3 describes the model.

The steady state distribution of the queuing network can be obtained from the BCMP theorem.

The Power of a queue is calculated for this model. The objective function as given by 4 is then optimized by the tuning algorithms.

6 Work to be Done

The queuing model that has been considered is a basic model and there is scope for modifying this model so as to mimic a real system better. Some areas that can be explored in this regard are

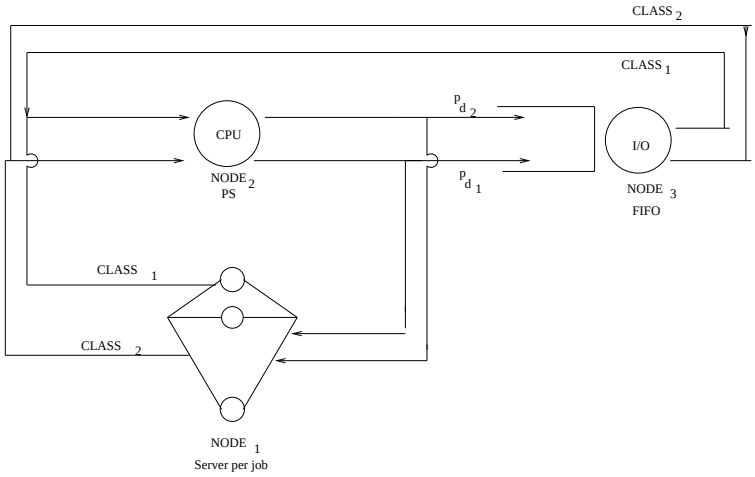

Figure 3: Our Queuing Model

- Priority Queue

  Every request coming is associated with a priority number and the service it receives from the system depends on this number. These systems can be classified into preemptive and non-preemptive systems.

- Balking Systems

  These are systems in which the customers choose to renege. Such customers do not contribute to the throughput, but utilize the system resources.

- Systems with correlated arrival streams

Such a model may offer a greater number of tunables that need to be varied. Thus the dimensionality of the objective function that needs to be optimized increases. In such a situation, it may not be straight forward to determine if the given function is convex.

The optimization techniques that have been considered for tuning could be improved for better performance. A more elaborate literature survey might give us more techniques for tuning.

7 Acknowledgment

This project has been a collaborative work with Sudheer Poojary. There is significant overlap with [9].

References

[1] Anderson, E. J., and Ferris, M. C. A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal on Optimization 11*, 3 (2000), 837–857.

[2] Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the Associatlon for Computing Machinery 22*, 2 (1975), 248–260.

[3] Elnaffar, S., Martin, P., and Horman, R. Automatically classifying database workloads. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management* (New York, NY, USA, 2002), ACM Press, pp. 622–624.

[4] Gelenbe, E., and Mitrani, I. *Analysis and Synthesis of Computer Systems*. Academic Press, 1980.

[5] Jain, R., Chiu, D. M., and Hawe, W. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Tech. Rep. DEC TR-301, Digital Equipment Corporation, Littleton, MA, 1984.

[6] Kiefer, J., and Wolfowitz, J. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics 23*, 3 (1952), 462–466.

[7] Kolias, C., and Kleinrock, L. The power function as a performance and comparison measure for ATM switches. *Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration. IEEE 1* (1998), 381–386 vol.1.

[8] Olsson, D. M., and Nelson, L. S. The Nelder-Mead simplex procedure for function minimization. *Technometrics 17*, 1 (1975), 45–51.

[9] Poojary, S. A study of self-tuning algorithms using queueing models. Master's thesis, IIT Bombay, 2008.

[10] Reiser, M., and Kobayashi, H. On the convolution algorithm for separable queuing networks. In *SIGMETRICS '76: Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation* (New York, NY, USA, 1976), ACM, pp. 109–117.

[11] Robbins, H., and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics 22*, 3 (1951), 400–407.