Load Balancing in an Heterogeneous Cluster

David Carrillo Cisneros*
Department of Computer Science
University of California, Irvine
Irvine, CA 92617
carrild1@uci.edu

Daniel Miller
University of California, Irvine
Irvine, CA 92617
dmiller@uci.edu

Abstract

Load balancing is a crucial part in every implementation of a cluster or grid computing environment. The election of an optimal strategy is bound by the characteristics of the architecture, the application of the computing environment, computational power, time constraints, and the nature of the information available about the system and its tasks. In this paper we propose a new load balancing strategy for the TREx environment [1] aimed to improve the execution time of the jobs in the system.

1 Introduction

1.1 Goal

The objective of this project is to find a load balancing strategy that improve the existing one in the TREx (Transparent Remote Execution) environment [1]. The current implementation of the load balancing module in such environment follows a greedy approach that assign each task to the server with the highest amount of computational power available. This approach does not consider the total set of tasks to execute in order perform the assignation. We believe that a more efficient assignation of tasks will considerably improve the performance of the TREx environment.

1.2 Previous Work

Load balancing is a typical problem in the field of distributed systems. Several efficient solutions have been proposed to several different architectures.

This work is based upon the work of the authors of [2] where they proposed a methodology to classify tasks accordingly to the resources the tasks will require to execute. In this paper the authors also proposed an iterative method to assign workload to the server. This method requires extensive communication with other terminals since for each iteration of the assignation loop it is required to update the local knowledge about the workload of other terminal. This imposes a delay in the assignation of the task that make it unfeasible for small tasks, specially in high latency networks, since the improvement gained by the distribution of the task is diluted by the time that takes for the task to be assigned. Additionally this method is heuristic, it is not guaranteed whether it provides an optimal assignation and its efficacy is only empirical.

Authors of [3] posed a similar problem to the one in this paper. They found an optimal load balancing solution using convex optimization techniques and then compared this strategy with other decentralized strategies. The model proposed by authors of [3] consider all the task to belong to the same class and does not consider the variance in execution time among them.

*dcarrill@ics.uci.edu

On [4], the authors present a model to estimate the service availability in a cluster, they propose probabilistic models to estimate this value and compare different strategies of back-up in case of failure of some of the machines. This models provide a useful insight about characteristics of estimation of execution time of requests, but do not provide an optimal load balancing strategy.

1.3 Approach

Our approach pose the assignation problem as a convex optimization problem, using only the information available through the TREX environment [1]. We model an estimation of the total time required by the system as a whole to finish the execution of a set of tasks and minimize it. The proposed model does not take into consideration the queue of a server which implies that the whole set of tasks will always to be contained in the queue of any server. Additionally, the model require independence between the rate of consumption of different resources within the same server and assume that the arrivals of task to process are independent of each other.

The proposed model relies in the classifier defined in [2], the adequacy of this classifier is not treated into this paper.

We proposed two model, both of them aim to assign static workload without requiring communication between the assignation unit and the surrogated computers. Although this approach prevents the assignation unit of obtaining useful information for the assignation, it allows the assignment to be performed in a extremely fast manner. We believe that this make it appropriate for small tasks in high delay networks (and commodity systems) where the cost of communication delay is significant when compared with the execution time of a task.

The first model proposed is an optimal static assignment. Upon observation of the behavior of this approach, we determined that the variance of the execution time could be reduced introducing a penalization value into the objective function, which was done in the second approach.

2 Methodology

2.1 Model Formulation

Consider a cluster of servers, all of them interconnected through a communication network. A load balancing unit has as job to assign tasks to each one of this servers. Each task can be classified as member of a class of tasks, this is shown in Figure 1.

The problem is modeled as follows:

S: The set of servers that will execute the tasks. Number of elements in S is denoted by $|S|$
T: The set of tasks to be executed. Number of elements in T is denoted by $|T|$
K: The set of distinct tasks classes. Number of elements in T is denoted by $|K|$

Each task $t \in T$ belongs to only one class $k \in K$. The mapping of task to classes is to be performed by a classifier which is out of the scope of this paper, previous work on this classification problem can be found on [2].

Each server $s \in S$ contains a variety of R distinct resources (CPU, GPU, Bandwidth, I/O communication). A server s is represented by a Performance Vector (P_s):

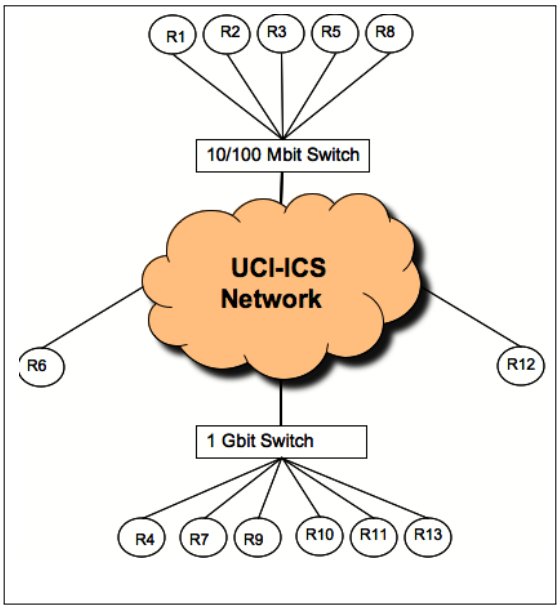

Figure 1: Representation of architecture.

$$P_s = \begin{bmatrix} P_{s,1} \\ P_{s,2} \\ \vdots \\ P_{s,r} \\ \vdots \\ P_{|S|,|R|} \end{bmatrix} \tag{1}$$

where each entry r in this vector represents the expected rate at which a resource r of server s will process a task assigned to it.

A class $k \in K$ is an approximate representation of the expected behavior of all the task that belongs to this class and is represented by the vector K_s:

$$K_s = \begin{bmatrix} T_{s,1} \\ T_{s,2} \\ \vdots \\ T_{s,r} \\ \vdots \\ T_{|S|,|R|} \end{bmatrix} \tag{2}$$

where each entry represents the expected the time taken to be consumed by a resource with unitary rate at resource r. It is assumed that the execution rates of different resources in the same server are independent.

Thus, the expected execution time of a task of class k when assigned to resource s is given by:

$$max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \tag{3}$$

Let $N_{s,k}$ be a random variable that represents the number of task of class k assigned to server i. Then, the Execution Time for a server s to finish its assigned load is given by:

$$\sum_k N_{s,k} max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \tag{4}$$

And its Expected Execution Time is:

$$E\left[\sum_k N_{s,k} max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \right] \tag{5}$$

Since $max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\}$ is a constant, the previous equation can be expressed as:

$$\sum_k E\left[N_{s,k}\right] max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \tag{6}$$

using basic properties of expected value.

The time for the whole system to finish its tasks load will be equal to the time taken by the last server to finish its own tasks load. Thus the Expected Execution Time for the system as a whole (the time taken to the system to process all its assigned workload) is given by:

$$max_s \left\{ \sum_k E\left[N_{s,k}\right] max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \right\} \tag{7}$$

2.2 Assignation Model

Define $Prob(S = s, K = k)$ as the probability distribution function of assigning a task of class k to a a server s. By conditional probability:

$$Prob(S = s|K = k) = \frac{Prob(S=s,K=k)}{Prob(K=k)}$$

$$\text{then } Prob(S = s, K = k) = Prob(S = s|K = k)Prob(K = k).$$

Define $M_k :=$ Number of tasks of class k to be assigned; this value is known since the tasks to assign are knew before the assignment is done. Assuming the distribution of classes to be independent, then:

$$Prob(K = k) = \frac{M_k}{|T|}$$

Then $E[N_{s,k}]$ can be expressed as:

$$E\left[N_{s,k}\right] = M_k Prob(S = s, K = k) = M_k Prob(S = s|K = k)Prob(K = k) \tag{8}$$

Defining $Prob(S = s|K = k)$ to follow a multinomial probability with probabilities $x_{1,k}, x_{2,k} \ldots x_{s,k}, \ldots, x_{|S|,k}$, then:

$$E\left[N_{s,k}\right] = M_k x_{s,k} Prob(K = k) \tag{9}$$

Thus, the Maximum Expected Execution time for the server is:

$$max_s \left\{ \sum_k M_k x_{s,k} Prob(K = k) max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \right\} \tag{10}$$

where both $M_k x_{s,k} Prob(K = k)$ and $max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\}$ are known constants which product can be expressed as $q_{s,k}$. Thus previous equation becomes:

$$max_s \left\{ \sum_k x_{s,k} q_{s,k} \right\} \tag{11}$$

2.3 Minimization of Execution Time

As our first approach, we attempt to minimize the Expected Execution Time given the model described above.

2.3.1 Optimization Problem

Equation 11 defines the objective function to minimize, during its formulation we restricted the values of $x_{s,k}$ to denote the conditional probability of a multinomial distribution function given k. In order for the values $x_{s,k}$'s to meet this requirement, the following constraints must be satisfied:

$$\begin{array}{lcll} \sum_s x_{s,k} & = & 1 & \forall k = 1, \ldots K \\ x_{s,k} & \geq & 0 & \forall s = 1, \ldots, S \text{ and } k = 1, \ldots, K \end{array}$$

To express the problem in a more condensed fashion, we define the following matrices:

$$X_s = \begin{bmatrix} x_{s,1} \\ \vdots \\ x_{s,|K|} \end{bmatrix}, \quad X = \begin{bmatrix} X_1 \\ \vdots \\ X_{|S|} \end{bmatrix}, \quad Q_s = \begin{bmatrix} q_{s,1} \\ \vdots \\ q_{s,|K|} \end{bmatrix} \tag{12}$$

Thus, the complete minimization problem can be expressed as:

$$minimize_X \; max_s \left\{ Q_s^T X_s \right\} \tag{13}$$

Subject to:

$$\begin{array}{lcll} -x_{s,k} & \leq & 0 & \forall \; s = 1, \ldots, |S| \text{ and } k = 1, \ldots, |K| \\ \sum_{s=1}^{|S|} x_{s,k} - 1 & = & 0 & \forall \; k = 1, \ldots |K| \end{array}$$

The objective function can be reformulated to remove the maximum function and obtain:

$$minimize_X \; t \tag{14}$$

Subject to:

$$\begin{array}{lcll} Q_s^T X_s - t & \leq & 0 & \forall \; s = 1, dots, |S| \\ -x_{s,k} & \leq & 0 & \forall \; s = 1, \ldots, |S| \text{ and } k = 1, \ldots, |K| \\ \sum_{s=1}^{|S|} x_{s,k} - 1 & = & 0 & \forall \; k = 1, \ldots |K| \end{array}$$

Which is a Linear Optimization Problem (LP).

2.3.2 Solution to the Optimization Problem

The Lagrangian of the minimization problem is:

$$L(x, \lambda, \nu) = t + \sum_{i=1}^{|S|} \lambda_i Q_i^T X_i - \sum_{i=1,j=1}^{|S|,|K|} \lambda_{(|S|+i)} x_{i,j} + \sum_{i=1}^{|K|} \nu_i \left(\sum_{j=1}^{|S|} x_{i,j} - 1 \right) \tag{15}$$

and applying the KKT conditions we obtain the following system of linear equations:

$$
\begin{aligned}
Q_i^T X_i &\leq 0 && (16)\\
-x_{i,j} &\leq 0 && (17)\\
\sum_{j=1}^{|S|} x_{i,j} - 1 &= 0 && (18)\\
\lambda_i &\geq 0 && (19)\\
\lambda_i Q_i^T X_i &= 0 && (20)\\
-\lambda_{(|S|i+j)} x_{i,j} &= 0 && (21)\\
\nabla t + \sum_{i=1}^{|S|} \lambda_i \nabla Q_i^T X_i - \sum_{i=1,j=1}^{|S|,|K|} \lambda_{(|S|+i)} \nabla x_{i,j} + \sum_{i=1}^{|K|} \nu_i \nabla \left(\sum_{j=1}^{|S|} x_{i,j} - 1 \right) &= 0 && (22)
\end{aligned}
$$

Equation 22 is equivalent to:

$$\sum_{i=1}^{|S|} \lambda_i Q_i - \sum_{i=1,j=1}^{|S|,|K|} \lambda_{(|S|+i)} + \sum_{i=1}^{|K|} \nu_i = 0 \tag{23}$$

This is a system of linear equations which solution (when exists) gives the values of X that minimize the Expected Process Time for the whole system.

2.4 Accounting for variance

As an attempt to reduce the variance in the Execution Time, a second approach is attempted. For this approach, a reformulation of the problem is made that takes into consideration the variance of the execution times across multiple runs.

Recall equation 4:

$$\sum_k N_{s,k} max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \tag{24}$$

that denotes the Execution Time for server s. And note that even tasks that belong to the same class will have a different Execution Time in the same server (the behavior of this difference will depend of the classification method), to model the previously constant value can be redefined to be a random variable:

$$Y_{s,k} = max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\} \tag{25}$$

which variance and estimated value can be known from the characteristic of the classifier used to classify the tasks. Thus Equation 4 becomes:

$$\sum_k N_{s,k} Y_{s,k} \tag{26}$$

and using known properties of variance of a product of random variables, its variance is:

$$Var(\sum_k N_{s,k} Y_{s,k}) = \sum_k \left(E[N_{s,k}]^2 Var(Y_{s,k}) + E[Y_{s,k}^2] Var(N_{s,k}) + Var(N_{s,k}) Var(Y_{s,k}) \right) \tag{27}$$

From section 2.4, $N_{s,k}$ follows a multinomial distribution, thus:

$$
\begin{aligned}
E[N_{s,k}] &= M_k x_{s,k} Prob(K=k) \\
Var[N_{s,k}] &= M_k x_{s,k} Prob(K=k)(1-(x_{s,k}))
\end{aligned}
$$

$$
\begin{aligned}
Var(\sum_k N_{s,k} Y_{s,k}) &= \sum_k ((M_k x_{s,k} Prob(K=k))^2 Var(Y_{s,k}) \\
&+ E[Y_{s,k}]^2 Prob(K=k) M_k x_{s,k}(1-x_{s,k}) \\
&+ M_k x_{s,k} Prob(K=k)(1-x_{s,k}) Var(Y_{s,k}))
\end{aligned}
$$

This variance equation can be used to generate an estimator of the variance of the Execution time of the whole system. A detailed analysis about the statistical characteristics of this estimator are beyond of the scope of this paper. Such estimator is:

$$
V(X_s) = Var(\sum_k N_{s,k} Y_{s,k}) \tag{28}
$$

Since $V(X)$ is a variances its value is always non-negative. We can prove that is a convex function using the criteria of the second derivate:

$$
\begin{aligned}
Var(\sum_k N_{s,k} Y_{s,k}) &= \sum_k \left((M_k x_{s,k})^2 Var(Y_{s,k}) + E[Y_{s,k}]^2 M_k x_{s,k}(1-x_{s,k}) + M_k x_{s,k}(1-x_{s,k}) Var(Y_{s,k})\right) \\
&= \sum_k \left((M_k x_{s,k})^2 (E[Y^2] - E[Y]^2) + E[Y_{s,k}]^2 M_k x_{s,k}(1-x_{s,k}) + M_k x_{s,k}(1-x_{s,k})(E[Y^2] - E[\right. \\
&= \sum_k \left(E[Y^2](M_k^2 x_{s,k}^2 + M_k x_{s,k}(1-x_{s,k})) + E[Y]^2 (M_k x_{s,k}(1-x_{s,k}) - (M_k^2 x_{s,k}^2)\right) \\
Var(\sum_k N_{s,k} Y_{s,k})' &= \sum_k \left(E[Y^2] M_k (M_k 2x_{s,k} - x_{s,k} + (1-x_{s,k})) + E[Y]^2 M_k (-x_{s,k} + (1-x_{s,k}) - (M_k 2x_{s,k})\right) \\
Var(\sum_k N_{s,k} Y_{s,k})'' &= \sum_k \left(E[Y^2] M_k (2M_k - 1 - 1) + E[Y]^2 M_k (-1 - 1 - 2M_k)\right) \\
&= \sum_k \left(2E[Y^2] M_k (M_k - 1) + 2E[Y]^2 M_k (-1 - M_k)\right)
\end{aligned}
$$

$$
\begin{aligned}
V(X)'' \geq 0 \iff & \sum_k \left(2E[Y^2] M_k (M_k - 1) + 2E[Y]^2 M_k (-1 - M_k)\right) \geq 0 \\
\rightarrow \quad & 2E[Y^2] M_k (M_k - 1) + 2E[Y]^2 T_k (-1 - M_k) \geq 0 \forall k \\
\rightarrow \quad & E[Y^2] \geq E[Y]^2 \forall k
\end{aligned}
$$

Since $Var(Y) = E[Y^2] - E[Y]^2 \geq 0$ then the inequality above is always true. Thus $V(X)'' \geq 0$ and $V(X)$ is convex.

Table 1: Overview of the characteristics of the resources used for application profiling.

Resource	Processor	Speed	Memory	Operating System
R1	Intel Core 2 Duo	2.4 GHz	3.0GB	XP Professional
R2	Intel Pentium 4	3.0 GHz	1GB	XP Professional
R3	AMD Athlon	1.0 GHz	512 MB	XP Professional
R4	Intel Core 2 Duo	2.2 GHz	1.5 GB	XP Professional

2.4.1 New Minimization Problem

Using the variance indicator derived in the previous section, a new minimization problem can be posed:

$$minimize_X\ t \tag{29}$$

Subject to:

$$\begin{array}{lcll} Q_s^T X_s + sV(X) - t & \leq & 0 & \forall\ s = 1, \ldots, |S| \\ -x_{s,k} & \leq & 0 & \forall\ s = 1, \ldots, |S| \text{ and } k = 1, \ldots, |K| \\ \sum_{s=1}^{|S|} x_{s,k} - 1 & = & 0 & \forall\ k = 1, \ldots, |K| \end{array}$$

where s is a parameter to be chosen from experimentation data.

3 Results

To compare between different approaches we will use the Expected Execution Time of the whole system (as described in previous section). The variance of this value will be a secondary criteria.

The application profile was obtained using four distinct resources shown in Table 1.

Benchmark techniques similar to the ones described in [5] and [6] where employed. This techniques used bytecode counting and timing similar to the methods used by the JRAF-2 framework and associated publications [7], [8]. This benchmark criteria is consistent to the one employed in [2].

Since the classifier described in [2] is used as departing point of this paper, the same routines were profiled for this paper, this routines are image processing extensions to the Python Interpreter of an application called ImagePack [9] which contains three possible class: FullPack, Half-Pack, and QuarterPack. Fullpack resize and converts two images to grayscale, enhances the sharpness of the images, difference them and write the result. HalfPack and QuarterPack perform the same operations but over a image which resolution is a half and one quarter of the original, respectively. The images utilized are described in Table 2.

The processing rates of the most taxed resources in each server (per task) are shown in table 3. This values are the ones denoted in equation 3 as $max_j \left\{ \frac{T_k[j]}{P_i[j]} \right\}$. The variances of Execution Time per resource and class are shown in table 4.

Both problems where solved utilizing IBM ILOG CPLEX Optimization Studio V12.2 [10] and modeled on "Optimization Programming Language" (OPL) [11]. The Appendix contain the OPL code of the model. The behavior of the solutions to the optimization problem were simulated in the "R Language"; the appendix contains the code using for this simulation. A complete implementation in TREx [1] is planned as part of future work.

Table 2: Images utilized for benchmark.

Image	Resolution	Pixels
Fallujah, Iraq	2094x1659	34687527
Fallujah, Iraq	4188x3317	13875011
Fallujah, Iraq	8366x6634	55500044
Indiana University	6500X4286	27859000
Istanbul, Turkey	3600x1998	7192800
Istanbul, Turkey	5116x2839	14524324
Istanbul, Turkey	11189x6208	69461312
Kabul, Afghanistan	1440x799	1150560
Kabul, Afghanistan	3600x1998	7192800
Kabul, Afghanistan	14198x7878	111851844
Mexico City, Mexico	1440x799	1150560
Mexico City, Mexico	7200x3995	28764000
Mexico City, Mexico	13320x7391	98448120
Purdue University	6500x4286	27859000
Riga, Latvia	5000x2699	13495000
San Clemente, CA	4300x2386	10259800
San Clemente, CA	12672x7031	89096832
UCI	14400x7990	230112000
Vatican City	28800x15980	460224000

Table 3: Value of most taxed resource.

Resource	FullPack	HalfPack	Quarter Pack
R1	11.50130	4.87316	2.86764
R2		16.9556243	9.25800
R3		50.33736	38.93386
R4	11.58038	4.82544	2.78167

3.1 Problem 1

The solution to the equation system posed in Section 2.3.2, when 20 task of class 1, 50 of class 2 and 80 of class 3 are to be assigned is given by Equation 30:

$$X = \begin{bmatrix} 0.97832 & 0 & 0 \\ 0 & 0 & 0.45576 \\ 0 & 0.13412 & 0 \\ 0.021684 & 0.86588 & 0.54424 \end{bmatrix} \tag{30}$$

The results of simulating this assignation code 300 times is shown in Figure 3.1. The mean Execution Time is 366.0147 s, its variance is 468.6658 s^2 and its STDDEV is 21.64869 s.

In order to allow the reader to appreciate the advantages of this approach against the previous greedy implementation, the previous load balancing unit on TREx[1] would have performed an assignation which mean execution time will be approximately 574.0918 s.

3.2 Problem 2

As described during in the Methodology section, Problem 2 requires a proper value of s to be calculated using experimental data. Since we lacked information about the behavior of the optimization problem as a function of s, our approach consisted in iterate over multiples of 0.1 in the interval $[0, 10]$ to observe the behavior of the function, while doing this, we observed that the variance of the Execution Time increased abruptly in the interval $[0, 0.1]$.

Table 4: Variances of execution times.

Resource	FullPack	HalfPack	Quarter Pack
R1	4.823	0.7316	0.18734
R2		1.506243	0.73502
R3		6.32796	2.33286
R4	3.49021	0.94767	0.2146

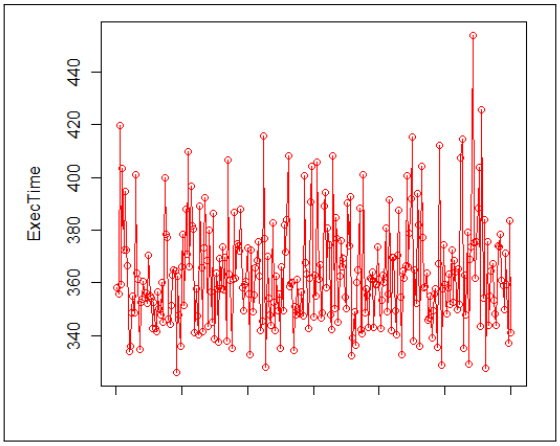

Figure 2: Model 1 Execution time (in second).

Further investigation about he behavior of the optimization problem is required before being able to derive a method to calculate an optimal value of s.

4 Discussion

Our first approach provided a considerable reduction in the Expected Execution Time of the over the one of the actual load balancing unit of the TREx system. Nevertheless is far from being optimal since does not takes in consideration important factors common to every distributed system environment as overload of transmission of tasks, bandwidth considerations during task distribution, and is not robust against outdated information. Additionally, is based upon several assumptions about the behavior of the tasks (i.e. as independence in rate of consumption of different resources in the same server) that are false and its efficiency is highly dependent of the quality of the classifier used to assign classes to the tasks to execute.

The second approach attempted to minimize this dependency of the classifier by introducing the variance in a class (a unwanted attribute of the classifier) as a penalization during the assignation of tasks. Unfortunately, simulation results that are contrary to what we expected prevent us to withdrawn conclusions about the efficacy (or inefficacy) of the approach until any programming or implementation error is discarded.

Further extensions over the same model are to be performed in the same model, as accounting for bandwidth and latency in the transmission and a penalization for outdated information.

During the realization of this project, we learned to model a typical problem of our research area as a convex optimization problem, to evaluate its solubility and to extend it to include new and different aspects of the problem of study.

5 Conclusion

In this paper we addressed the problem of load balancing when the architecture does not provide timely information about the instant state of the components of a grid. We developed a linear pro-

gramming model for the load balancing problem and derived a metric, the Expected Execution Time, to estimate the efficacy of a load balancing strategy and solved the problem of finding an optimal solution for this metric. Additionally, we extended this model to account for the variance in the execution time and formulated this new problem as a Quadratically Constrained Optimization Problem. Simulations were conducted to evaluate the performance of this models and evaluate the feasibility of its implementation in TREx[1].

References

[1]Richert Wang, Enrique Cauich, and Isaac D. Scherson. Federated clusters using the transparent remote Execution (TREx) environment. Parallel and Distributed Systems, International Conference on, 2:18, 2007.

[2] John U. Duselis, E. Enrique Cauich, Richert K. Wang, Isaac D. Scherson. Resource Selection and Allocation for Dynamic Adaptive Computing in Heterogeneous Clusters CLUSTER 2009: 1-10

[3] E. Altman, U. Ayesta, B.J. Prabhu. Load Balancing in Processor Sharing Systems ValueTools '08 Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools

[4] Chee-Wei Ang, Chen-Khong Tham. Analysis and optimization of service availability in a HA cluster with load-dependent machine availability IEEE Transactions on Parallel and Distributed Systems, September 2007 (vol. 18 no. 9)

[5] Umesh Krishnaswamy and Isaac D. Scherson. A Framework for Computer Performance Evaluation Using Benchmark Sets. IEEE Transactions on Computers, 49(12):13251338, 2000.

[6] Walter Binder and Jarle Hulaas. Exact and Portable Profiling for the JVM Using Bytecode Instruction Counting. Electronic Notes in Theoretical Computer Science, 164(3):45 64, 2006. Proceedings of the 4th International Workshop on Quantitative Aspects of Programming Languages (QAPL 2006), Quantitative Aspects of Programming Languages 2006.

[7] Walter Binder and Jarle Hulaas. Using Bytecode Instruction Counting as Portable CPU Consumption Metric. Electronic Notes in Theoretical Computer Science, 153(2):57 77, 2006. Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005).

[8] David L. Wangerin and Isaac D. Scherson. Using predictive adaptive parallelism to address portability and irregularity. Parallel Architectures, Algorithms, and Networks, International Symposium on, 0:370 377, 2005.

[9] Nicholas Carriero, Eric Freeman, David Gelernter, and David Kaminsky. Adaptive Parallelism and Piranha. Computer, 28(1):4049, 1995.

[10] IBM ILOG CPLEX Optimization Studio (V12.2) [Software] (2007).

[11] Optimization Programming Language (OPL) [Software]

[12] R Language [Software]

Appendix

OPL implementation of Approach 1

```
dvar float x1[1..3] in 0..1;
dvar float x2[2..3] in 0..1;
dvar float x3[2..3] in 0..1;
dvar float x4[1..3] in 0..1;
dvar float t in 0..4000000;

range k_range = 1..3;

int t1=30;
int t2=50;
int t3=80;
int t_array[k_range] = [t1, t2, t3];

float r1[k_range] = [11.5013, 4.87316, 2.86764];

float r2[2..3] = [16.9556243, 9.25800];

float r3[2..3] = [50.33736,38.93386];

float r4[k_range] = [11.58038, 4.82544,2.78167];

minimize
  t;

subject to {
  // Server 1
  ct1:  sum(i in 1..3)r1[i]*t_array[i]*x1[i]<= t;

  // Server 2
  ct2: sum(i in 2..3)r2[i]*t_array[i]*x2[i]<= t;

  // Server 3
  ct3:  sum(i in 2..3)r3[i]*t_array[i]*x3[i]<= t;
  // Server 4
  ct4:  sum(i in 1..3)r4[i]*t_array[i]*x4[i]<= t;


  // Assignation for every class must be a probability
  ct9: x1[1]+x4[1]  == 1;
  ct10: x1[2]+x2[2]+x3[2]+x4[2]  == 1;
  ct11: x1[3]+x2[3]+x3[3]+x4[3] == 1;
}
```

OPL implementation of Approach 2

```
dvar float x1[1..3] in 0..1;
dvar float x2[2..3] in 0..1;
dvar float x3[2..3] in 0..1;
dvar float x4[1..3] in 0..1;
dvar float t in 0..4000000;

range k_range = 1..3;

int t1=30;
int t2=50;
int t3=80;
int t_array[k_range] = [t1, t2, t3];
```

```
float p[k_range] = [(t1+t2+t3)/t1, (t1+t2+t3)/t2, (t1+t2+t3)/t3];

float r1[k_range] = [11.5013, 4.87316, 2.86764];

float r2[2..3] = [16.9556243, 9.25800];

float r3[2..3] = [50.33736,38.93386];

float r4[k_range] = [11.58038, 4.82544,2.78167];

// Variances
float v1[k_range]= [4.823,0.7316,0.18734];
float v2[2..3]= [1.506243, 0.73502];
float v3[2..3]= [6.32796 , 12.33286];
float v4[k_range]= [3.49021,0.94767, 0.2146];

// Parameter
float s = 0.1;


minimize
  t;

subject to {
  // Server 1
  ct1:  sum(i in 1..3)r1[i]*t_array[i]*x1[i]
   + s*(sum(i in 1..3)( (t_array[i]*x1[i])^2 + (r1[i]^2 + v1[i])*(t_array[i]*x1[i])*(1-x1[i]*p
<= t;

  // Server 2
  ct2: sum(i in 2..3)r2[i]*t_array[i]*x2[i]
   + s*(sum(i in 2..3)( (t_array[i]*x2[i])^2 + (r2[i]^2 + v2[i])*(t_array[i]*x2[i])*(1-x2[i]*p
<= t;

  // Server 3
  ct3:  sum(i in 2..3)r3[i]*t_array[i]*x3[i]
   + s*(sum(i in 2..3)( (t_array[i]*x3[i])^2 + (r3[i]^2 + v3[i])*(t_array[i]*x3[i])*(1-x3[i]*p
<= t;

  // Server 4
  ct4:  sum(i in 1..3)r4[i]*t_array[i]*x4[i]
   + s*(sum(i in 1..3)( (t_array[i]*x4[i])^2 + (r4[i]^2 + v4[i])*(t_array[i]*x4[i])*(1-x4[i]*p
<= t;


  // Assignation for every class must be a probability
  ct9: x1[1]+x4[1]  == 1;
  ct10: x1[2]+x2[2]+x3[2]+x4[2]  == 1;
  ct11: x1[3]+x2[3]+x3[3]+x4[3] == 1;
}
```

R Implementation of Simulation

```
reps = 300
ExecTime = rep(0,reps);

t1=30;
t2=50;
t3=80;

r1=c(11.5013, 4.87316, 2.86764);
r2=c(0, 16.9556243, 9.25800);
r3=c(0, 0.33736,38.93386);
r4=c(11.58038, 4.82544,2.78167);
```

```
v1=c(4.823,0.7316,0.18734);
v2=c(0,0,01.506243, 0.73502);
v3=c(0, 6.32796 , 12.33286);
v4=c(3.49021,0.94767, 0.2146);

for(iter in c(1:reps)){


x = matrix(c(
c(0.97832, 0, 0),
c(0, 0, 0.45576),
c(0, 0.13412, 0),
c(0.021684, 0.86588, 0.54424)),
4, byrow=T);

assigns = matrix(c(
rmultinom(1, t1, x[,1]),
rmultinom(1, t2, x[,2]),
rmultinom(1, t3, x[,3])), 3, byrow=T);

tasks1<-mat.or.vec(4,1)
tasks2<-mat.or.vec(4,1)
tasks3<-mat.or.vec(4,1)

server1=numeric(0);
server2=numeric(0);
server3=numeric(0);
server4=numeric(0);

for(k in c(1:3)){
server1 = c(server1, rnorm(assigns[k,1], r1[k], v1[k]));
server2 = c(server2, rnorm(assigns[k,2], r2[k], v2[k]));
server3 = c(server3, rnorm(assigns[k,3], r3[k], v3[k]));
server4 = c(server4, rnorm(assigns[k,4], r4[k], v4[k]));
}

timeS1= sum(server1);
timeS2= sum(server2);
timeS3= sum(server3);
timeS4= sum(server4);

ExecTime[iter] = max(timeS1, timeS2, timeS3, timeS4);
}
mean(ExecTime)
sd(ExecTime)
```