Optimization of Job Assignment on a Computational Grid

Daniel Vanderster

M. A. Sc. Student

Department of E.C.E

University of Victoria

August 10, 2003

Abstract

This paper presents a method for the minimization of total completion time of a set of computational processes distributed across a number of geographically distributed processor clusters. First, the simple case with only a single processor at each site is solved. Next, the minimization is solved for the case when total completion time at each cluster is non-linearly dependent on the number of processes assigned. Finally, the minimization is solved when a constraint on the total number of assigned processes is added.

Contents

List of Figures

List of Tables

1 Introduction

1.1 What is Grid Computing?

Anyone who has heard of projects such as SETI@Home[1], or Genome@Home[2], will have a general idea of Grid computing. These projects utilize the spare central processing unit (CPU) cycles on personal computers in a coordinated effort to solve a very large problem. The relatively small contributions from thousands of processors have resulted in the creation of a virtual supercomputer much more powerful than any one system. Distributed applications such as these have shown the possibilities of large scale computing using commodity CPUs. The next step is to create a distributed system, or Grid, where all kinds of computing resources can be shared across administrative domains. The computational Grid owes its name to the electrical power grid which allows people to tap into a valuable resource. The computing Grid will similarly open up immense resources in which supercomputers, processor farms, disks, databases, informatic systems, collaborative tools, and people are linked by a high speed network.

1.2 Assigning Resources for Jobs

When coordinating resources which are distributed over thousands of kilometers, it can be difficult to achieve performance results which are similar to those from closely knit resources. The Grid must allocate resources such that the time to complete applications is minimized. This task will include the customization of Grid-aware applications, as well as the development of high-performance networks to connect remote resources. However, most important to the minimization of total completion time is the ratio of job assignment to each site. For this reason, this paper seeks to find the optimum number of processes assigned to each cluster in the Grid, given some performance characteristics about each cluster.

The type of application that can be distributed in the method discussed here is very

[1] The Search for Extra-Terrestrial Intelligence, found at http://setiathome.ssl.berkeley.edu/

[2] http://genomeathome.stanford.edu

specific. These applications, usually called *embarrassingly parallel*, can be sent to arbitrary processors without the need to communicate with any other process. The experiments carried out by the applications are usually of the *parametric sweep* type. These experiments execute the same binary program at each processor, varying the parameters given to the application at each processor.

2 Formulation of the Problem

2.1 Case 1: Single Processor at Each Site

The first case to be solved is that where each distributed site contains only a single processor. Thus, the problem at hand is to run N identical jobs on M different processors. The objective is to find the job assignment set

$$\mathbf{n} = [n_1 \quad n_2 \quad \cdots \quad n_M]^T \tag{1}$$

where n_i is the number of jobs assigned to the i-th processor. Assume that the time to execute one job at the i-th site is c_i – the set of such run-times is

$$\mathbf{c} = [c_1 \quad c_2 \quad \cdots \quad c_M]^T \tag{2}$$

When executed on a single processor, the jobs cannot be run simultaneously. Thus total execution time at the i-th site is

$$T_i = c_i n_i \tag{3}$$

Example 1 *Take for example the following situation. Let us assume that there are 3 participating sites and we would like to run our set of 50 jobs. The times to execute a single job at each site are 2 hours, 4 hours, and 7 hours.*

In this example we are therefore given the following quantities, $N = 50$, $M = 3$, *and* $\mathbf{c} = [2 \quad 4 \quad 7]^T$. *The objective is to find the job assignment set* $\mathbf{n} = [n_1 \quad n_2 \quad n_3]^T$.

2.1.1 Analytical Solution to Case 1

For the first case of this problem, there exists an analytical solution which is rather simple to find. In order to find the job assignment set $\mathbf{n}$, it is clear that the time to be minimized is the overall runtime T, where $T = \max_i |T_i|$. So, the problem becomes

$$minimize \quad \max_i |T_i(n_i)| \tag{4}$$

Now we inspect the case where the overall runtime T will be minimized. Using the numbers from Example 1, we can infer some results about the minimization in the general case. One idea is to run all the jobs on the fastest processor – this would result in a runtime $T = T_1 = 50 * 2 = 100$. However, if we move a single job from the 1st to the 2nd processor, the resulting runtime would be smaller, $T = \max(T_1, T_2) = T_1 = 49 * 2 = 98$. We continue this procedure until the limiting case, when $T = T_1 = T_2 = T_3$. At this point, the we have achieved the minimum runtime T. In general, the minimum runtime T occurs when

$$T_1(n_1) = T_2(n_2) = \cdots = T_M(n_M) \tag{5}$$

From this relationship we can derive a system of equations which will result in the optimized set $\mathbf{n}$. This system is

$$T_1 - T_2 = 0 \tag{6}$$

$$T_1 - T_3 = 0 \tag{7}$$

$$\vdots$$

$$T_1 - T_M = 0 \tag{8}$$

$$T_2 - T_3 = 0 \tag{9}$$

$$\vdots$$

$$T_2 - T_M = 0 \tag{10}$$

$$\vdots$$

$$T_{M-1} - T_M = 0 \tag{11}$$

$$n_1 + n_2 + \cdots + n_M = N \tag{12}$$

As you can see, this system contains $M+1$ equations. Thus, one of the first M equations is linearly dependent on the others. It is dropped and each T_i is replaced with $c_i n_i$ to achieve

$$\begin{bmatrix} c_1 & -c_2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_1 & 0 & 0 & \cdots & 0 & -c_M \\ 0 & c_2 & -c_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & c_{M-2} & 0 & -c_M \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ \vdots \\ n_{M-1} \\ n_M \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ M \end{bmatrix} \tag{13}$$

Solving this system of linear equations results in the job assignment set **n**.

Example 2 *Using the system in Equation 13, we can solve the problem given in Example 1. With the parameters given, we have*

$$\begin{bmatrix} 2 & -4 & 0 \\ 2 & 0 & -7 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 50 \end{bmatrix} \tag{14}$$

This can be reduced using Matlab to the following

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} = \begin{bmatrix} 28 \\ 14 \\ 8 \end{bmatrix} \tag{15}$$

so the job assignment $\mathbf{n} = [28 \quad 14 \quad 8]^T$. *Thus, this set of jobs will take* $T = 28*2 = 14*4 = 8*7 = 56$ *hours to complete.*

2.2 Case 2: Non-Linearly Dependant Total Execution Time

2.2.1 Problem Generalization Using the p-Norm

In order to solve the minimization in cases where T_i is not linearly dependent on n_i, we first note that Equation 4 will be satisfied when the difference between all T_i's is minimized. In

other words, the problem can be expressed as:

$$minimize \quad \max_i |T_i(n_i) - T_j(n_j)| \quad \text{for all } i, j \text{ where } i \neq j \tag{16}$$

In order to find a method to express this objective function analytically, we use the infinity-norm [3]

$$||\mathbf{x}||_\infty \equiv \max_i |x_i| \tag{17}$$

which can be expressed as the limit of the p-norm, or

$$\lim_{p\to\infty} ||\mathbf{x}||_p = ||\mathbf{x}||_\infty \tag{18}$$

with

$$||\mathbf{x}||_p \equiv \left(\sum_i |x_i|^p\right)^{1/p} \tag{19}$$

Therefore, with a significantly large value for p, we can say that:

$$\max_i |x_i| \approx \left(\sum_i |x_i|^p\right)^{1/p} \tag{20}$$

So the objective function for minimization is generalized as:

$$T(\mathbf{n}) = \sqrt[p]{[T_1(n_1) - T_2(n_2)]^p + [T_1(n_1) - T_3(n_3)]^p + \cdots + [T_{M-1}(n_{M-1}) - T_M(n_M)]^p} \tag{21}$$

2.2.2 Elimination of Constraints

In order to solve some problems which will become apparent later, let us assume that each cluster must be assigned at least one job to execute. Thus, we want

$$n_i \geq 1 \quad \forall\ i \quad \text{or}$$
$$n_i - 1 \geq 0 \tag{22}$$

If we vary u_i instead of n_i, with the property

$$n_i = u_i^2 + 1 \tag{23}$$

then the constraint is satisfied. Thus, the objective function becomes:

$$T(\mathbf{u}) = \sqrt[p]{[T_1(u_1^2+1) - T_2(u_2^2+1)]^p + \cdots + \left[T_{M-1}(u_{M-1}^2+1) - T_M(u_M^2+1)\right]^p} \tag{24}$$

3 Optimization Using Computational Methods

3.1 Case 1: Minimizing the P-Norm for the Single CPU Case

Having an analytic objective function, given in Equation 24, the problem can now be solved numerically. The solution is constructed in the source code given in Appendix 1, Section 1. First, the objective function is created as `ff.m`. The gradient of this function is not evaluated analytically, rather numerically by dividing the function's change over a small distance δ. `ff.m` uses three other functions, `T1.m`, `T2.m`, and `T3.m` to represent the run-time functions at each site.

The minimization is carried out using two quasi-Newton methods, the Davidon-Fletcher-Powell (DFP) method, and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. These functions are implemented in `dfp.m` and `bfgs.m`. (These are slightly modified versions of code made available by [2]). Each of these functions carries out a line search using the Fletcher's Line Search (FLS) implemented in `fls.m` [2].

The minimization is initiated by calling the Matlab script in `run.m`. This script carries out a few steps to find the solution to the problem. First, initial conditions are set: in this case $\mathbf{u}_0 = [10 \;\; 10 \;\; 10]^T$ and $\epsilon = 10^{-5}$. The script then runs the DFP algorithm. The solution $\mathbf{u}^*$ is then converted to $\mathbf{n}^*$ by Equation 23. This $\mathbf{n}^*$ is not necessarily equal to the analytical results achieved in the previous section. However, the ratios between the number of jobs assigned to each site are the same as previously. Thus, the results can be normalized to $N = 50$ by multiplying $\mathbf{n}^*$ by $\frac{50}{||\mathbf{n}^*||_1}$. The results of the computation for both methods are given in Table 1.

3.1.1 Discussion of Simple Case Results

As shown in Table 1, the results achieved using the p-norm are equivalent to those achieved analytically. We also see that the DFP algorithm finds the solution in 14 iterations, while the BFGS algorithm takes 24. Through a variety of tests, it became apparent that the resulting $\mathbf{u}^*$ is highly dependent on the initial point $\mathbf{u}_0$. More about this property was discovered

Method	$\mathbf{u}^*$	$\mathbf{n}^*$	Normalized $\mathbf{n}^*$	$T(Norm.\mathbf{n}^*)$	Iterations
DFP	$\begin{bmatrix} 1.5965 \\ 0.8800 \\ 0.1179 \end{bmatrix}$	$\begin{bmatrix} 3.5487 \\ 1.7743 \\ 1.0139 \end{bmatrix}$	$\begin{bmatrix} 28.0000 \\ 14.0000 \\ 8.0000 \end{bmatrix}$	$\begin{bmatrix} 56.0000 \\ 56.0001 \\ 55.9999 \end{bmatrix}$	14
BFGS	$\begin{bmatrix} 1.7671 \\ 1.0302 \\ 0.4218 \end{bmatrix}$	$\begin{bmatrix} 4.1226 \\ 2.0613 \\ 1.1779 \end{bmatrix}$	$\begin{bmatrix} 28.0000 \\ 14.0000 \\ 8.0000 \end{bmatrix}$	$\begin{bmatrix} 55.9999 \\ 56.0001 \\ 56.0001 \end{bmatrix}$	24

Table 1: Minimization Results for the Case 1. These results are equivalent to the results achieved analytically.

when solving the problem in the non-linear case.

3.2 Case 2: Minimizing the P-Norm with Non-Linear Run-Times

3.2.1 Finding the Non-Linear Run-Times

The simple situation described and solved in Case 1 is in fact not realistic in the Grid environment. Most sites involved in a Grid usually share between 10 and 100 processors for use by the Grid community. Also, in general, the total completion time $T_i(n_i)$ at a single site does not scale linearly with the number of jobs n_i assigned. In most cases, processors on a cluster share central resources, such as file servers and network switches. Thus, even though a cluster may have x CPUs, the cluster cannot complete x jobs in the same time as one job.

This behaviour is demonstrated in Figure 1, which shows the total completion time of the BaBar Monte Carlo Simulation (a particle physics application run at UVic) versus the number of simultaneous jobs. When 4 jobs are run, the completion time is 4.5 hours, but when 12 jobs are run, the time increases to 5.75 hours. In fact, the completion time T seems to be exponentially dependent on the number of jobs submitted. To demonstrate this fact, one can take the natural logarithm of the run-times T and then fit a 1^{st} order function to

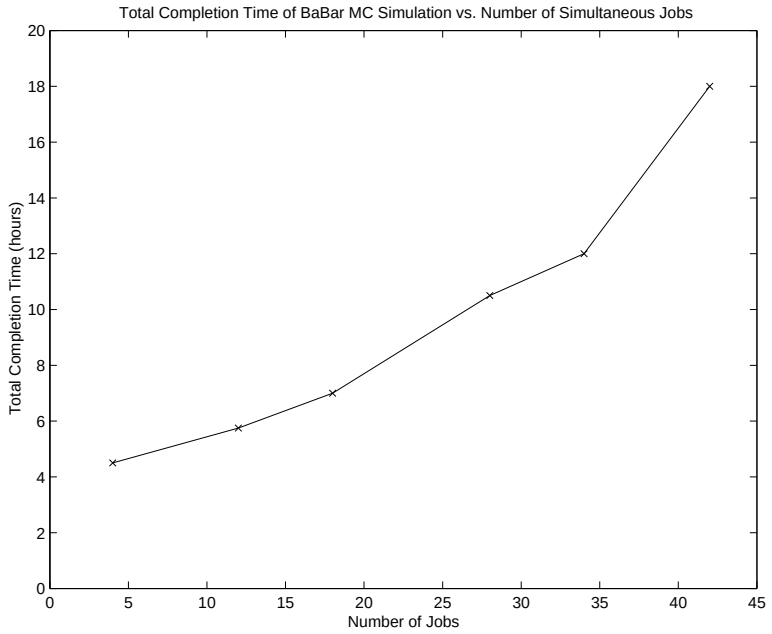

Figure 1: Plot showing total completion time of BaBar Monte Carlo Simulation versus number of simultaneous jobs.

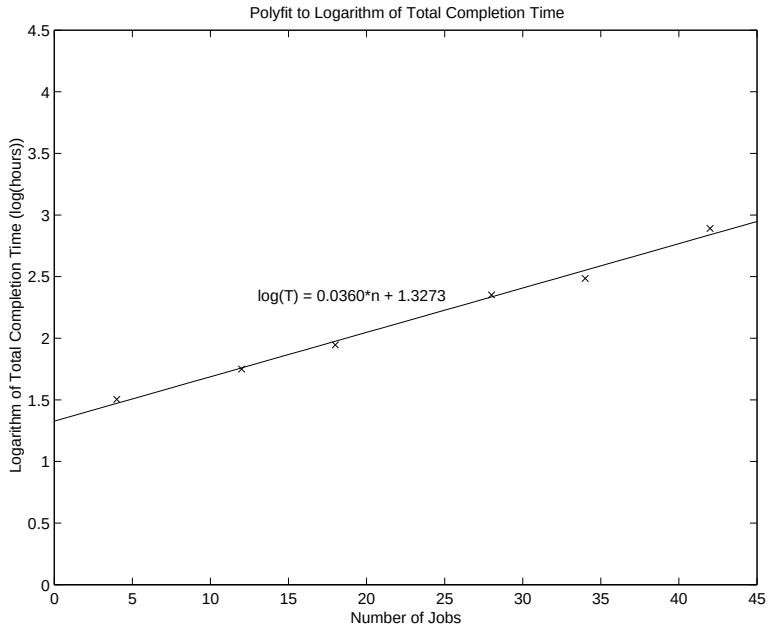

Figure 2: Plot showing experimental data on a log scale.

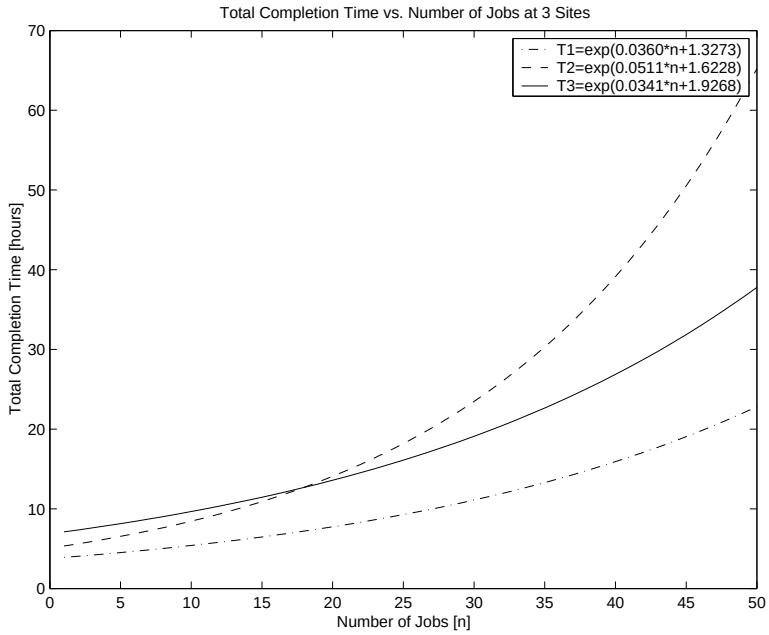

Figure 3: Plot showing functions to be used in non-linear minimization.

the resulting data. This was done, as shown in Figure 2, and the resulting line was

$$\log(T_1) = 0.0360n + 1.3273 \tag{25}$$

For the purposes of this project, two additional characteristic equations were selected:

$$\log(T_2) = 0.0511 * n + 1.6228 \tag{26}$$

$$\log(T_3) = 0.0341 * n + 1.9268 \tag{27}$$

The one flaw with using these exponential equations to represent T_i at each site is that the predicted run-time at $n_i = 0$ is non-zero. This is a non-realistic, as we hardly expect a cluster to take time to execute zero jobs. Thus, we have the constraint that $n_i \geq 1$ for all i. A plot of Equations 25, 26, and 27 is shown in Figure 3.

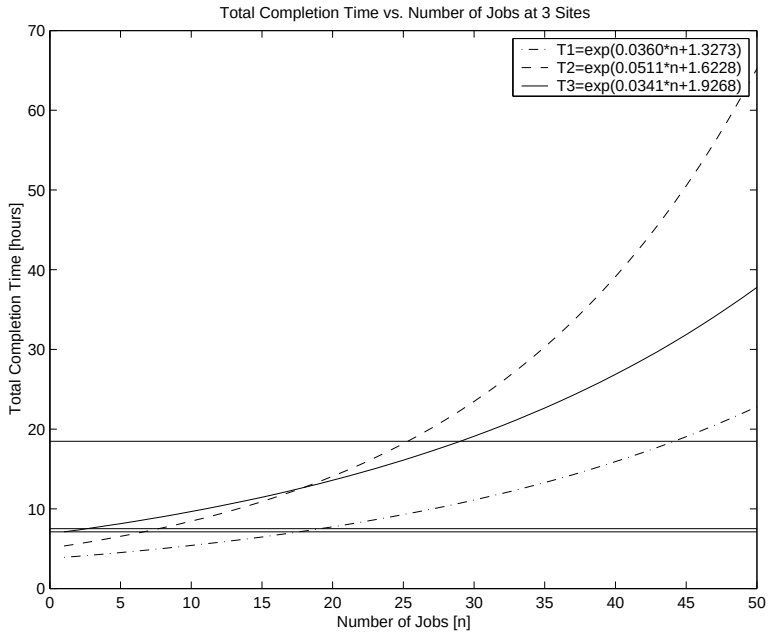

Figure 4: Plot showing solutions to non-linear case.

3.2.2 Minimizing the P-Norm for the Non-Linear Case without the N Constraint

In order to solve the minimization in case 2, the code from case 1 was modified slightly (this code is given in Appendix 1, Section 2). Files `T1.m`, `T2.m`, and `T3.m` were modified to represent Equations 25, 26, and 27. A number of initial points $\mathbf{u}_0$ were used and the results are given in Table 2.

3.2.3 Discussion of Case 2 Results

Interpretation of the results in case 2 yields a few interesting points. First, more evidently than in case 1, we see that the optimal job assignment set $\mathbf{n}^*$ is dependent on the initial point $\mathbf{u}_0$. Whereas this problem was solved in case 1 by scaling the result, this cannot be done with the non-linear functions of cases 2 and 3. This property would cause quite a problem for a job scheduler, since the sum of the number of jobs, $||\mathbf{n}^*||_1$, is not necessarily equal to N, and it is difficult to control this value using $\mathbf{u}_0$.

Method	$\mathbf{u}_0$	$\mathbf{u}^*$	$\mathbf{n}^*$	$T(\mathbf{n}^*)$	Iterations
DFP	$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 4.1120 \\ 2.4152 \\ -0.5705 \end{bmatrix}$	$\begin{bmatrix} 17.9083 \\ 6.8336 \\ 1.3254 \end{bmatrix}$	$\begin{bmatrix} 7.1850 \\ 7.1850 \\ 7.1850 \end{bmatrix}$	16
BFGS	$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 4.0806 \\ 2.3775 \\ -0.2327 \end{bmatrix}$	$\begin{bmatrix} 17.6513 \\ 6.6526 \\ 1.0542 \end{bmatrix}$	$\begin{bmatrix} 7.1189 \\ 7.1189 \\ 7.1189 \end{bmatrix}$	24
DFP	$\begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix}$	$\begin{bmatrix} 4.2537 \\ 2.5825 \\ 1.2559 \end{bmatrix}$	$\begin{bmatrix} 19.0941 \\ 7.6691 \\ 2.5774 \end{bmatrix}$	$\begin{bmatrix} 7.4984 \\ 7.4984 \\ 7.4984 \end{bmatrix}$	18
BFGS	$\begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix}$	$\begin{bmatrix} 4.2590 \\ 2.5885 \\ 1.2746 \end{bmatrix}$	$\begin{bmatrix} 19.1389 \\ 7.7006 \\ 2.6246 \end{bmatrix}$	$\begin{bmatrix} 7.5105 \\ 7.5105 \\ 7.5105 \end{bmatrix}$	21
DFP	$\begin{bmatrix} 7.5 \\ 7.5 \\ 7.5 \end{bmatrix}$	$\begin{bmatrix} 6.5667 \\ 4.9296 \\ 5.2914 \end{bmatrix}$	$\begin{bmatrix} 44.1213 \\ 25.3007 \\ 28.9991 \end{bmatrix}$	$\begin{bmatrix} 18.4611 \\ 18.4611 \\ 18.4612 \end{bmatrix}$	11
BFGS	$\begin{bmatrix} 7.5 \\ 7.5 \\ 7.5 \end{bmatrix}$	$\begin{bmatrix} 6.5679 \\ 4.9307 \\ 5.2930 \end{bmatrix}$	$\begin{bmatrix} 44.1371 \\ 25.3119 \\ 29.0157 \end{bmatrix}$	$\begin{bmatrix} 18.4716 \\ 18.4716 \\ 18.4716 \end{bmatrix}$	13
DFP	$\begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$	$\begin{bmatrix} 10.3638 \\ 8.3421 \\ 9.7913 \end{bmatrix}$	$\begin{bmatrix} 108.4089 \\ 70.5914 \\ 96.8686 \end{bmatrix}$	$\begin{bmatrix} 186.7966 \\ 186.7967 \\ 186.7963 \end{bmatrix}$	8
BFGS	$\begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$	$\begin{bmatrix} 10.3638 \\ 8.3421 \\ 9.7913 \end{bmatrix}$	$\begin{bmatrix} 108.4089 \\ 70.5914 \\ 96.8686 \end{bmatrix}$	$\begin{bmatrix} 186.7968 \\ 186.7970 \\ 186.7966 \end{bmatrix}$	8

Table 2: Minimization Results for the Non-Linear Case

To try to understand what is happening, three of the solutions, when $T^* = 7.1189$, $T^* = 7.5105$, and $T^* = 18.4716$ are plotted in Figure 4. Equation 5 explains that the solution occurs where $T_1 = T_2 = T_3$, thus intersection between any horizontal line and the functions in Figure 3 is a possible solution. This results in infinitely many solutions. In fact, the selection of $\mathbf{u}_0$ causes the algorithms to select the horizontal line which is nearest (as defined by each algorithm) to $\mathbf{u}_0$. Thus, an additional constraint must be used to specify that $||\mathbf{n}^*||_1$ must equal N, the desired number of submitted jobs. The method for adding this constraint and its effect on the solution is discussed in case 3.

3.3 Case 3: Minimizing the P-Norm with Non-Linear Run-Times and the N Constraint

3.3.1 Adding the N Constraint

The previous method discussed resulted in infinitely many solutions, all of which dependent on the initial point $\mathbf{u}_0$. In order to remove this dependency, the following constraint was added:

$$||\mathbf{n}||_1 = n_1 + n_2 + \cdots + n_M = N. \tag{28}$$

To incorporate this constraint to our problem, the Matlab code was modified to vary only n_1 and n_2, while $n_3 = N - n_1 - n_2$. The resulting code is given in Appendix 1, Section 3. Leaving the run-time functions the same as in case 2, the optimization algorithms were run and the results are given in Table 3.

3.3.2 Discussion of Results of Non-Linear Case with the N Constraint

As is shown in Table 3, the addition of the N constraint gives an optimal solution which is both independent of $\mathbf{u}_0$ and results in $N = 50$. Using this constraint, we have essentially forced the algorithm to select the single horizontal line (shown in Figure 5 whose intersecting function values sum to 50. Thus, the problem is solved, and the optimal $\mathbf{n}^*$, given three

Method	$\mathbf{u}_0$	$\mathbf{u}^*$	$\mathbf{n}^*$	$\sum n_i$	$T(\mathbf{n}^*)$	Iterations
DFP	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 5.0576 \\ 3.4557 \end{bmatrix}$	$\begin{bmatrix} 26.5789 \\ 12.9420 \\ 10.4791 \end{bmatrix}$	50	$\begin{bmatrix} 9.8172 \\ 9.8172 \\ 9.8172 \end{bmatrix}$	12
BFGS	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 5.0575 \\ 3.4557 \end{bmatrix}$	$\begin{bmatrix} 26.5788 \\ 12.9421 \\ 10.4791 \end{bmatrix}$	50	$\begin{bmatrix} 9.8172 \\ 9.8173 \\ 9.8172 \end{bmatrix}$	24
DFP	$\begin{bmatrix} 10 \\ 10 \end{bmatrix}$	$\begin{bmatrix} -5.0575 \\ 3.4557 \end{bmatrix}$	$\begin{bmatrix} 26.5788 \\ 12.9420 \\ 10.4792 \end{bmatrix}$	50	$\begin{bmatrix} 9.8172 \\ 9.8172 \\ 9.8172 \end{bmatrix}$	13
BFGS	$\begin{bmatrix} 10 \\ 10 \end{bmatrix}$	$\begin{bmatrix} -5.0575 \\ 3.4557 \end{bmatrix}$	$\begin{bmatrix} 26.5788 \\ 12.9421 \\ 10.4791 \end{bmatrix}$	50	$\begin{bmatrix} 9.8172 \\ 9.8172 \\ 9.8172 \end{bmatrix}$	17

Table 3: Minimization Results for the Non-Linear Case with the N Constraint

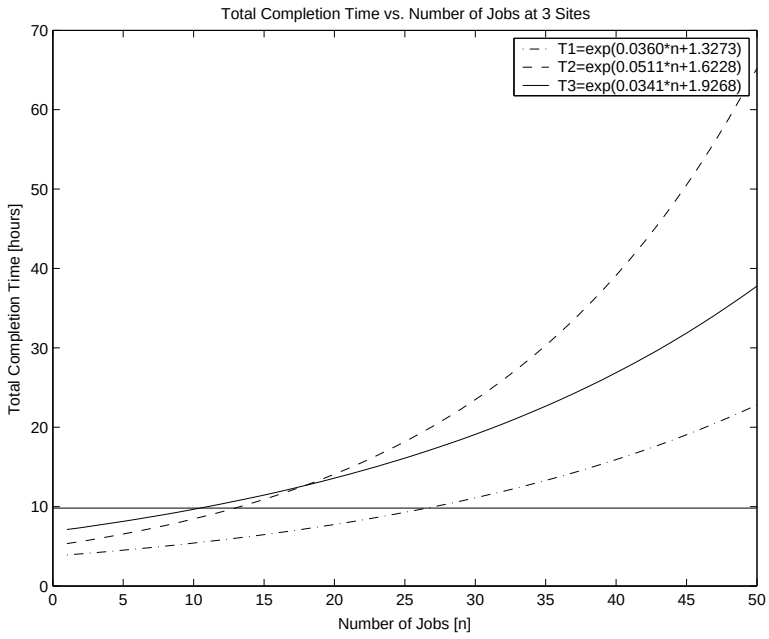

Figure 5: Plot showing solution to non-linear case with N constraint.

clusters with characteristic equations T_1, T_2, and T_3, is

$$\mathbf{n}^* = [26.5788 \quad 12.9421 \quad 10.4791]^T \tag{29}$$

Evaluating this solution with T_1, T_2, and T_3 results in the total completion time of 9.182 hours.

4 Conclusion

This paper has provided a method for the optimization of job assignment on a computational Grid. The optimum assignment is determined to be the one which minimizes the difference between the total completion time at each site in a Grid. This property is used to:

1. Solve a simple case of the problem, where each site has only a single processor, using a system of linear equations.

2. By minimizing the p-norm, solve the case where the run-time at each site is non-linearly dependent on the number of jobs assigned.

3. By minimizing the p-norm with a constraint on the total number of jobs submitted, solve the non-linearly dependent case.

Finally, this project has proven to be a challenging exercise in numerical optimization techniques, and its results will prove useful in the development of the Canadian national computational Grid.

References

[1] Antoniou, A. *Optimization: Theory and Practice*. Dept. of Electrical and Computer Engineering, University of Victoria. March 2000.

[2] Lu, W. S. ELEC 503 Website. http://www.ece.uvic.ca/~wslu/403/index.html. Dept. of Electrical and Computer Engineering, University of Victoria. March 2000.

[3] Weisstein, Eric W. "Vector Norm". *Eric Weisstein's World of Mathematics.* http://mathworld.wolfram.com/VectorNorm.html.

[4] Agarwal, Ashok. Notes on BaBar Run-times. 2003.

5 Appendix: MATLAB Code

5.1 Case 1: Single CPU at Each Site

5.1.1 run.m

```
x0 = [10 10 10]';
epsi = 1e-5;

disp('DFP:')
us = dfp(x0,epsi);
ns = us.^2+1
norm_ns = ns*50/norm(ns,1)
T1(norm_ns(1))
T2(norm_ns(2))
T3(norm_ns(3))

disp('BFGS:')
us = bfgs(x0,epsi);
ns = us.^2+1
norm_ns = ns*50/norm(ns,1)
T1(norm_ns(1))
T2(norm_ns(2))
T3(norm_ns(3))
```

5.1.2 ff.m

```
function [ y ] = ff( u )
p = 70;
u = u(:);
u1 = u(1);
u2 = u(2);
u3 = u(3);
y = ((T1(u1^2+1)-T2(u2^2+1))^p + (T1(u1^2+1)-T3(u3^2+1))^p +
    (T2(u2^2+1)-T3(u3^2+1))^p)^(1/p);
```

5.1.3 fg.m

```
function [ z ] = fg( x )
n = length(x);
dt = 1e-9;
f0 = ff(x);
I = eye(n);
for i=1:n
    z(i) = (ff(x+dt*I(:,i))-f0)/dt;
end
```

```
z = z(:);
```

5.1.4 T1.m

```
function [ y ] = f1( x )
x = x(:);
y = 2*x;
```

5.1.5 T2.m

```
function [ y ] = f1( x )
x = x(:);
y = 4*x;
```

5.1.6 T3.m

```
function [ y ] = f1( x )
x = x(:);
y = 7*x;
```

5.1.7 dfp.m

```
% dfp.m
% To apply the Davidon-Fletcher-Powell (DFP)
% quasi-Newton algorithm
function x = dfp(x0,epsi)

k = 0;
x = x0(:);
S = eye(length(x));
g = fg(x);
er = norm(g);
while er > epsi,
  d = -S*g;
  a = fls(x,d,'ff','fg');
  dx = a*d;
  xnew = x + dx;
  g1 = fg(xnew);
  ga = g1 - g;
  pp = S*ga;
  S = S + (dx*dx')/(dx'*ga) - (pp*pp')/(ga'*pp);
  x = xnew;
  g = g1;
  k = k + 1;
  er = norm(dx);
```

```
end
disp('optimal point:')
x
disp('number of iterations:')
k
```

5.1.8 bfgs.m

```
% bfgs.m
function x = bfgs(x0,epsi)

k = 0;
x = x0(:);
S = eye(length(x));
g = fg(x);
er = norm(g);
while er > epsi,
  d = -S*g;
  a = fls(x,d,'ff','fg');
  dx = a*d;
  xnew = x + dx;
  g1 = fg(xnew);
  ga = g1 - g;
  v1 = S*ga;
  v2 = ga'*dx;
  v3 = v1*dx';
  S = S +((v2+ga'*v1)/(v2^2))*(dx*dx') - (v3+v3')/v2;
  x = xnew;
  g = g1;
  k = k + 1;
  er = norm(dx);
end
disp('optimal point:')
x
disp('number of iterations:')
k
```

5.1.9 fls.m

```
function alpha = fls(xk,s,F,G,p1,p2)

% This m-file implements Fletcher's line search as described in
% Algorithm 7.4 of Dr. Antoniou's Notes "Optimization: Theory
```

```
% and Practice", and was written by John Wong, July 2001.
% Last modified: July 31, 2001.

% Usage: fls(x,s,F,G,p1,p2)
%    x:  Initial point
%    s:  Search direction
%    F:  Function to be minimized along the direction of s
%    G:  Gradient of function F
%    p1: Internal parameters that are required for the implementation of
%        the line search regardless of the application at hand.
%        It is a string (e.g. 'rho=.1') and can be a combination several
%        internal parameters (e.g., 'rho=.25;sigma=0.5').
%    p2: User-defined parameter vector. Note that p2 must be a VECTOR
%        with all components numerically specified. The order in which
%        the components of p2 appear must be the same as what they appear
%        in function F and gradient G. For example, if p2 = [a b], then
%        F.m and G.m must be in the form of
%        function z = F(x,p2)
%        and
%        function z = G(x,p2)
%    Useful p1's include:                                  default value
%        'rho=   ' defines right bracket                        0.1
%        'sigma= ' defines left bracket  (sigma >= rho)         0.1
%        'tau= '   defines minimum step for sectioning          0.1
%        'gamma= '                                              0.75
%    Discussion on the line search algorithm and selection of suitable
%    values of the parameters in p1 can be found in Chapter 7, Sections
%    10 and 11 of the Lecture Notes.
%    Example 1: where non-default values for some internal parameters
%               are specified:
%          alpha = fls(x,s,'myfun','mygrad','rho=0.25;sigma=0.5',[a b]);
%               where a and b are numerically specified parameters.
%    Example 2: where all internal parameters use their default values:
%               alpha = fls(x,s,'myfun','mygrad',[a b]);
%               where a and b are numerically specified parameters.
%    Example 3: where all internal parameters assume default values
%               and no user defined parameters:
%               alpha = fls(x,s,'myfun','mygrad');

k = 0;
m = 0;
tau = 0.1;
gamma = 0.75;
rho = 0.1;
```

```
sigma = 0.1;
mhat = 400;
epsilon = 1e-10;
xk = xk(:);
s = s(:);
parameterstring ='';
% evaluate given parameters:
  if nargin > 4,
    if isstr(p1),
      eval([p1 ';']);
    else
      parameterstring = ',p1';
    end
  end
  if nargin > 5,
    if isstr(p2),
      eval([p2 ';']);
    else
      parameterstring = ',p2';
    end
  end
% compute f0 and g0
  eval(['f0 = ' F '(xk' parameterstring ');']);
  eval(['gk = ' G '(xk' parameterstring ');']);
  m = m+2;
  deltaf0 = f0;
% step 2 Initialize line search
  dk = s;
  aL = 0;
  aU = 1e99;
  fL = f0;
  dfL = gk'*dk;
  if abs(dfL) > epsilon,
    a0 = -2*deltaf0/dfL;
  else
    a0 = 1;
 end
 if ((a0 <= 1e-9)|(a0 > 1)),
    a0 = 1;
 end
%step 3
 while 1,
    deltak = a0*dk;
    eval(['f0 = ' F '(xk+deltak' parameterstring ');']);
```

```
    m = m + 1;
%step 4
    if ((f0 > (fL + rho*(a0 - aL)*dfL)) & (abs(fL - f0) > epsilon) &
    (m < mhat))
      if (a0 < aU)
        aU = a0;
      end
      % compute a0hat using equation 7.65
      a0hat = aL + ((a0 - aL)^2*dfL)/(2*(fL - f0 + (a0 - aL)*dfL));
      a0Lhat = aL + tau*(aU - aL);
      if (a0hat < a0Lhat)
        a0hat = a0Lhat;
      end
      a0Uhat = aU - tau*(aU - aL);
      if (a0hat > a0Uhat)
        a0hat = a0Uhat;
      end
      a0 = a0hat;
    else
      eval(['gtemp =' G '(xk+a0*dk' parameterstring ');']);
      df0 = gtemp'*dk;
      m = m + 1;
      % step 6
      if (((df0 < sigma*dfL) & (abs(fL - f0) > epsilon) & (m < mhat) &
      (dfL ~= df0)))
        deltaa0 = (a0 - aL)*df0/(dfL - df0);
        if (deltaa0 <= 0)
          a0hat = 2*a0;
        else
          a0hat = a0 + deltaa0;
        end
        a0Uhat = a0 + gamma*(aU - a0);
        if (a0hat > a0Uhat)
          a0hat = a0Uhat;
        end
        aL = a0;
        a0 = a0hat;
        fL = f0;
        dfL = df0;
      else
        break;
      end
    end
 end % while 1
```

```
if a0 < 1e-5,
   alpha = 1e-5;
else
   alpha = a0;
end
```

5.2 Case 2: Multiple CPUs at Each Site, Without N Constraint

5.2.1 run.m

```
u0 = [10 10 10]';
epsi = 1e-5;

disp('DFP:')
us = dfp(u0,epsi);
ns = us.^2+1
T1(ns(1))
T2(ns(2))
T3(ns(3))

disp('BFGS:')
us = bfgs(u0,epsi);
ns = us.^2+1
T1(ns(1))
T2(ns(2))
T3(ns(3))
```

5.2.2 ff.m

```
function [ y ] = ff( u )
p = 50;
u = u(:);
u1 = u(1);
u2 = u(2);
u3 = u(3);
y = ((T1(u1^2+1)-T2(u2^2+1))^p + (T1(u1^2+1)-T3(u3^2+1))^p +
    (T2(u2^2+1)-T3(u3^2+1))^p)^(1/p);
```

5.2.3 fg.m

```
function [ z ] = fg( x )
n = length(x);
dt = 1e-10;
f0 = ff(x);
I = eye(n);
```

```
for i=1:n
    z(i) = (ff(x+dt*I(:,i))-f0)/dt;
end
z = z(:);
```

5.2.4 T1.m

```
function [ y ] = T1( n )
n = n(:);
%xs = [4 12 18 28 34 42]';
%ts = [4.5 5.75 7 10.5 12 18]';
%p = polyfit(xs,log(ts),1);

y = exp(polyval([0.0360 1.3273],n));
```

5.2.5 T2.m

```
function [ y ] = T2( n )
n = n(:);
y = exp(polyval([0.0511 1.6228],n));
```

5.2.6 T3.m

```
function [ y ] = T3( n )
n = n(:);
y = exp(polyval([0.0341 1.9268],n));
```

5.3 Case 3: Multiple CPUs at Each Site, With N Constraint

5.3.1 run.m

```
u0 = [10 10]';
epsi = 1e-5;

disp('DFP:')
us = dfp(u0,epsi);
n1 = us(1)^2+1
n2 = us(2)^2+1
n3 = 50-n1-n2

disp('BFGS:')
us = bfgs(u0,epsi);
n1 = us(1)^2+1
n2 = us(2)^2+1
n3 = 50-n1-n2
```

5.3.2 ff.m

```
function [ y ] = ff( u )
p = 50;
u = u(:);
n1 = u(1)^2+1;
n2 = u(2)^2+1;
n3 = 50 - n1 - n2;
y = ((T1(n1)-T2(n2))^p + (T1(n1)-T3(n3))^p +
    (T2(n2)-T3(n3))^p)^(1/p);
```