# A Cost-Benefit Framework for Online Management of a Metacomputing System

Yair Amir, Baruch Awerbuch, R. Sean Borgstrom
Department of Computer Science
The Johns Hopkins University
Baltimore MD 21218
{yairamir, baruch, rsean}@cs.jhu.edu

## 1. ABSTRACT

**Managing a large collection of networked machines, with a series of incoming jobs, requires that the jobs be assigned to machines wisely. A new approach to this problem is presented, inspired by economic principles: the Cost-Benefit Framework. This framework simplifies complex assignment and admission control decisions, and performs well in practice. We demonstrate this framework in two different environments: an Internet-wide market for computational services and the classic network of workstations.**

### 1.1 Keywords

Networks, resource allocation, metacomputing, markets.

## 2. INTRODUCTION

Collections of networked machines are common in the modern world. Using each individual machine as a completely independent computer is obviously inefficient – one machine could be working on a dozen jobs while the others sit idle. A metacomputing system is a set of networked machines that can pool their computational resources to avoid this problem. Each machine has several computational resources associated with it. Intelligent management of a metacomputing system requires a good strategy for assigning computational resources to jobs the system must perform.

Computational resources are heterogeneous. A given task may require a certain amount of normalized CPU seconds, Megabytes of memory, and network bandwidth. Not only are these resources independent, they are not even directly comparable – they are measured in unrelated units. This can make it difficult to determine the optimal machine to which to assign a given computational task. This situation can be further complicated by tasks with different priority levels and different requirements in terms of completion time. More generally, tasks can have different rewards that the system will receive for completing them.

The Cost-Benefit framework is a unified approach to these two problems, inspired by economic principles. The approach is straightforward: we assign a homogeneous *cost* to each resource, which depends on that resource's current utilization. The total cost for all resources used is the cost of a given scenario. The *marginal cost* method for assigning jobs puts a job on the machine where its resource consumption has the minimum marginal cost. The resulting scenario has the minimum total cost. Jobs with different priorities and demands are assigned a homogeneous *benefit*, which depends on the time in which that job is completed. Benefits and costs are measured in the same units. The optimal course for a system using the Cost-Benefit framework is to maximize the profit of the system – the total benefit earned minus the costs paid. (Minimizing the costs improves system efficiency; maximizing the benefit directly improves the system's value to its users.)

This relatively simple concept turns out to have useful theoretical properties. In particular, when certain exponential cost functions are used, the marginal cost strategy is competitive – its consumption of resources is at most O(log n) greater than that of an optimal strategy that knows the future (where n is the number of machines with resources). This strategy, in effect, can predict a system's future needs regardless of the correlation between past and future.

It is important to emphasize that the Cost-Benefit framework is not intended to help a metacomputing system reach a steady state. Instead, it is designed to maximize the system's transient performance. The framework is not an “offline” optimization problem where all the variables are known in advance; it must manage the system “online,” making decisions based on the system’s current state and online demands at any given time.

In this paper, we study how the Cost-Benefit framework helps manage online metacomputing systems. Our test case is the *Java Market*, a computational market designed for use with this framework. The Java Market “buys” spare cycles from interested machines anywhere on the Internet and “sells” their computational services to anyone on the Internet with a job written in Java. The concepts of costs, benefits, and negotiations between the two are fundamental to its design.

To demonstrate the generality of the Cost-Benefit framework, we also summarize our experience with it in another setting – a classic network of workstations. In this context, *Enhanced Mosix* (and its cousin *Enhanced PVM*) are strategies for job allocation and reallocation in a scalable computing cluster. We have developed these strategies in a joint work with the Mosix group [1, 2, 3, 4].

Each of these test cases has been shown to produce very good performance in practice. The Java Market, using machines connected only by the Internet, produced a dramatic increase in our sample application's speed. Because the machines used only Web browsers to connect to the Market, they could have been anywhere in the world, running any operating system -- and the machines would still have been secure. Similarly, the Enhanced PVM and Enhanced Mosix strategies proved to be much more efficient than standard resource allocation methods on a local cluster, where both our strategies and the traditional strategies had complete control over the machines.

## 3. RELATED WORK

The LYDIA project [5] studies single-resource resource allocation on a system where there are many kinds of jobs, and each "class" of jobs has different performance expectations. The performance of each class is given a homogeneous cost called the "performance index", much like our job benefit functions. The LYDIA project does not yet address the complex issues involved in balancing several of the diverse system resources simultaneously.

SPAWN [6] and other systems like it provide computational markets where tasks bid competitively for resources. This approach uses economic principles conceptually similar to ours. The Cost-Benefit framework, however, integrates the computer science notions of competitive algorithms and analysis with the economics concepts of marginal costs and markets for services.

The Mosix system [2, 4] enhances several operating systems with the ability to transparently move jobs from one machine to another. This allows the system to quickly correct all serious mistakes in job allocation. Mosix is a powerful system independently and can be combined with our resource allocation strategies for even stronger performance.

The Condor system [7] is similar to the Java Market, one of our testbeds. The Condor team has created a set of software tools for utilizing the wasted CPU cycles in a cluster of workstations. Condor provides a checkpointing mechanism for the jobs it schedules, allowing interrupted jobs to be resumed later when a machine of the appropriate architecture is available. Although Condor is a mature system, proven to work efficiently with hundreds of machines at a time, it has very limited support for heterogeneous machine architectures. The Java Market, while limited by the speed of the virtual Java machines available, is 100% cross-platform.

The Popcorn project [8], independently developed at the Hebrew University in Israel, is an online market for computational services that shares many features with the Java Market. The project differs from the Java Market primarily in that it provides a new programming model. Its users must write their applications with the Popcorn project in mind -- Popcorn cannot be used with ordinary Java applications. A key Java Market design decision is that users submit their jobs as regular Java applications, and the Market itself does all necessary modifications.

Milan [9] and Javelin [10], like the Popcorn project, provide programming models that take advantage of heterogeneous Internet-connected machines. Also like the Popcorn project, these systems are not designed for use with standard Java applications, and their resource allocation strategies are not provably competitive with the optimal strategy.

## 4. THE COST-BENEFIT FRAMEWORK

We will use the same conceptual structure for the computational market and the metacomputing system that we will study. In both cases, we will examine the system in terms of *benefits* and *costs*, where the system tries to maximize its benefit and minimize its cost.

### 4.1 Cost

The key to our Cost-Benefit framework is that the cost of a resource is an exponential function of its utilization. Each time a certain amount (e.g. 10%) of the resource is used, the cost for that resource doubles. In its simplest form, this could be used to perform admission control: if the benefit of a job is higher than its cost, the job is admitted. (See Figure 4.1.1).

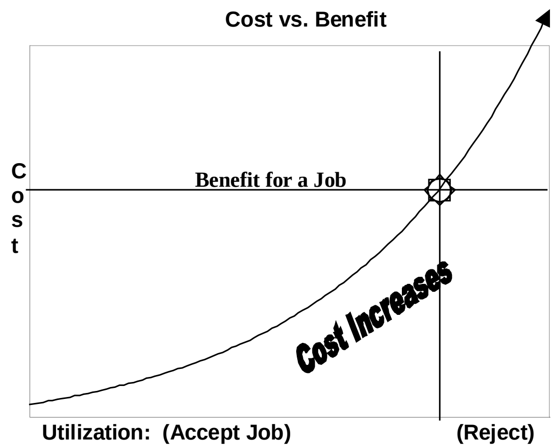


**Figure 4.1.1: Exponential Costs**

The picture becomes more interesting when we have multiple resources that can be exhausted. Normally, decision-making in this circumstance is difficult – but with the Cost-Benefit model, we can simply add these costs together. This makes decision-making very easy. For example, suppose we have two machines. A job comes in for one of these machines requiring *m* Megabytes of memory and *c* normalized seconds of CPU time. The cost for this job's memory usage will be different on each machine, based on how much memory is already used. Similarly, the cost to share the CPU will be different on each machine, based on how many jobs are already using that machine. We place the job on the machine where the sum of these marginal costs is minimized. (See Figure 4.1.2).

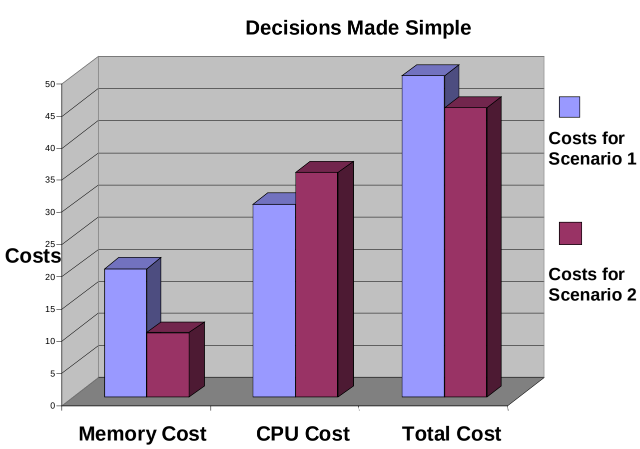


**Figure 4.1.2 Our Cost Function Makes Choosing the Right Machine Simple!**

Our framework's "costs" and real-world costs like poor machine performance are related as follows. Using our cost function when performing admission control and job assignment provides an upper bound on these real-world costs - and has proved to reduce them dramatically in practice. (See section 4.3.)

## 4.2 Benefit

The other half of the Cost-Benefit framework is the benefit earned from jobs. This benefit can be compared to the marginal cost for running a job to see whether running the job on a given machine is worthwhile. If the job cannot make a "profit" on any machine, then the job is rejected or delayed, depending on the system.

In the simplest case, the benefit for completing a job is equal to its priority, much as in Figure 4.1.1. Jobs with low benefit will be rejected or delayed when the system is heavily loaded. Jobs with high benefit are more likely to be admitted immediately. A job whose benefit is greater than the highest possible cost will always be accepted and placed on a machine.

Things become more interesting when jobs have benefit *functions*. For example, a job might give no benefit at all unless it is completed within a given time frame, or its benefit might reduce linearly over time. (See Figure 4.2.1.) The benefit for a job might even become negative, if enough time passes, meaning that the party that submitted the job must be compensated for lost time.

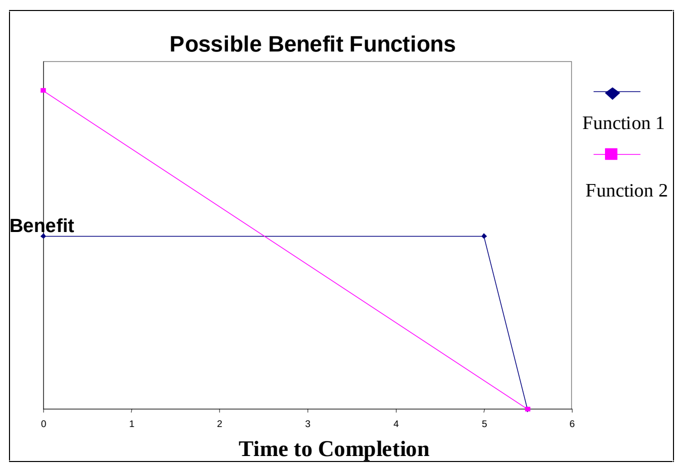


**Figure 4.2.1: Benefit as a Function of Time Taken**

When benefit functions are variable, as seen above, the system has a more complicated decision to make. Faster machines will complete jobs more rapidly, earning more benefit, but the more jobs are running on a given machine, the higher its cost will be. In this context, the job and the system will have to "negotiate" to determine where to place the job.

## 4.3 Cost Function

One particular cost function, used with our Cost-Benefit framework, shows very nice theoretical properties. This cost function, in a cluster of *n* machines, charges:

$$n^{(usage\ /\ maximum\ usage)}$$

for each resource.

Using the marginal cost strategy with this particular cost function has a beneficial theoretical property. Over the system's continuously operating lifetime, the maximum usage of each resource is within O(log *n*) of the *optimal* assignment strategy's maximum usage [11]. Further, this holds even when the optimal strategy knows the future.

This theoretical guarantee is weak, but most job assignment strategies have no theoretical guarantees at all. Further, this particular strategy has been shown in tests to perform extremely well in practice. (See Section 5.)

Note that the maximum usage of a resource is not defined in terms of our abstract costs. Restricting these maxima translates directly into bounded CPU loads, bounded memory usage, limited network congestion, *etc.*

## 4.4 A Decentralized Approach

The Cost-Benefit framework uses a centralized scheduler in both of our test-beds. Experience with the Condor system [7] shows that one scheduler can manage hundreds of producer machines, as long as it acts only as a decision-maker.

For a more distributed approach, our framework can be extended as follows. Assign each scheduler a subset of the producer machines, which it must choose from when assigning jobs. These subsets can intersect, and they can be changed on-line. Clients can deliver their job to any scheduler, or even shop among schedulers for the lowest price. Assuming that the optimal assignment strategy is restricted to the same clients and the same producer subsets, the theoretical guarantee above will still apply.

## 5. THE JAVA MARKET

The Java Market is an Internet-wide market for computational services that is being developed at the Johns Hopkins University. It is the first test-bed built for our Cost-Benefit framework. (For more details, see http://www.cnds.jhu.edu/publications, technical report CNDS-98-1.)

## 5.1 The Model

The Java Market's world is defined by two entities: machines and jobs. Both machines and jobs contract with the Java Market: one sells computational services to the Market and one buys such services from the Market.

### *5.1.1 Jobs*

Each job *j* submitted to the system is defined by the following properties:

- Its arrival time, *a(j)*,
- Its resource vector, *r(j)*,
- The benefit function for this job, *b(j,t)*, where *t* is the time the job takes to complete.

The resource vector *r(j)* represents the various system resources the job requires to complete. In the current implementation of the Java Market, these resources are CPU speed and network connectivity. As the power of the Java language grows, it will become feasible to add resources such as memory and disk I/O to this model.

The benefit *b(j,t)* is the reward the system receives for completing job *j* in *t* time. In the Java Market, this is literally the amount the consumer of computational resources will pay to have their job completed in that time. It might be measured in terms of virtual money, usable to buy Java Market services, or real currency. For example, a consumer might be willing to pay $20 to complete a large simulation in 6 hours or less.

We will assume that only one job can be run on a given machine. The cost to run a job on a machine is computed using the exponential function of Section 4.3. This cost will then be converted to virtual or real money using observed trends in what people will sell their machines for and buy resources for. This allows the Java Market to weigh benefits and costs as the Cost-Benefit framework suggests.

### *5.1.2 Machines*

Machines can be *dedicated machines*, which guarantee their availability to the system for a certain length of time, or *opportunistic machines*, which offer unreliable service for an indefinite period of time.

Each machine *m* that is made available to the system is defined by the following properties:

- Its arrival time, *at(m)*,
- Its resource vector, *r(m)*,
- Its departure time, *dp(m)*,
- (For dedicated machines) its total cost, *c(m)*, and
- (For opportunistic machines) its cost per second of use, *cps(m)*.

The arrival time *at(m)* is the time that the machine signals its availability to the Market. The departure time *dp(m)*, similarly, is the time when the machine will signal that it is no longer available or break off its connection with the

Market. (e.g. when the machine's owner returns and removes the machine from the Market.)

The resource vector *r(m)* describes the relevant resources associated with the machine. These are, naturally, the same resources that are associated with jobs.

When a dedicated machine *d* is offered to the system, all of these properties are known. This is not a bidding system, and *c(d)* is assumed to be the "true" cost for *d*. The system must either accept *d* and pay the cost *c(d)* or reject *d* and pay nothing. If it accepts that machine, it can use *d*'s computational resources until time *dp(d)*.

When an opportunistic machine *o* is offered to the system, all of these properties except the departure time *dp(o)* are known. No immediate decision is necessary. At any point before the (unknown) departure time, the system can use *o*'s computational resources. If it completes a *t*-second job on machine *o*, it must pay the cost *t * cps(o)*. In other words, it must pay for each second of successful use.

In the Java Market, the costs represented by *c(m)* and *cps(m)* are literally paid to the machines used. This may be virtual money, usable only to buy the Java Market's services, or it may be real currency. For example, the Market might pay a provider $10 to use its machine for 8 hours while the operator is away.

Naturally, the Java Market cannot expect its providers to charge based on the cost model described in Section 3. However, it can determine whether to buy a dedicated machine's time and whether to use an opportunistic machine's resources using the value calculations described above.

## 5.2 The System

The Java Market is intended to be a true Internet-wide market for computational services. As illustrated in Figure 5.2.1, the Java Market is designed to transfer jobs from *any* machine on the Internet *to* any machine on the Internet that wishes to participate. There is no installation or platform-dependent code – the only requirement is that the jobs be written in Java. Further, the Java job does not need to be written especially for the Market -- the Market can rewrite Java applications into Applets automatically, and provides services that can overcome some of the inherent Applet restrictions. These applications can then be ported automatically to any producer machine. Using the Market is only slightly more difficult than clicking on a browser bookmark.

The Java Market has no dependence on any given architecture. Producers and consumers can be running any kind of machine, and any operating system, that has a Java-capable Web browser. In other words, it can handle heterogeneous machines as easily as the Cost-Benefit framework handles heterogeneous resources. The program-transfer technology associated with the Market has already been implemented. It is currently being developed into a true computational market and a test-bed for the Cost-Benefit framework.

The Java Market is implemented as a 5,000-line Java application. Roughly 2,000 lines represent the Market itself; 2,000 are libraries added to consumer applications after they are submitted; and 1,000 are the various Market applets and interfaces for those applets in the main program.

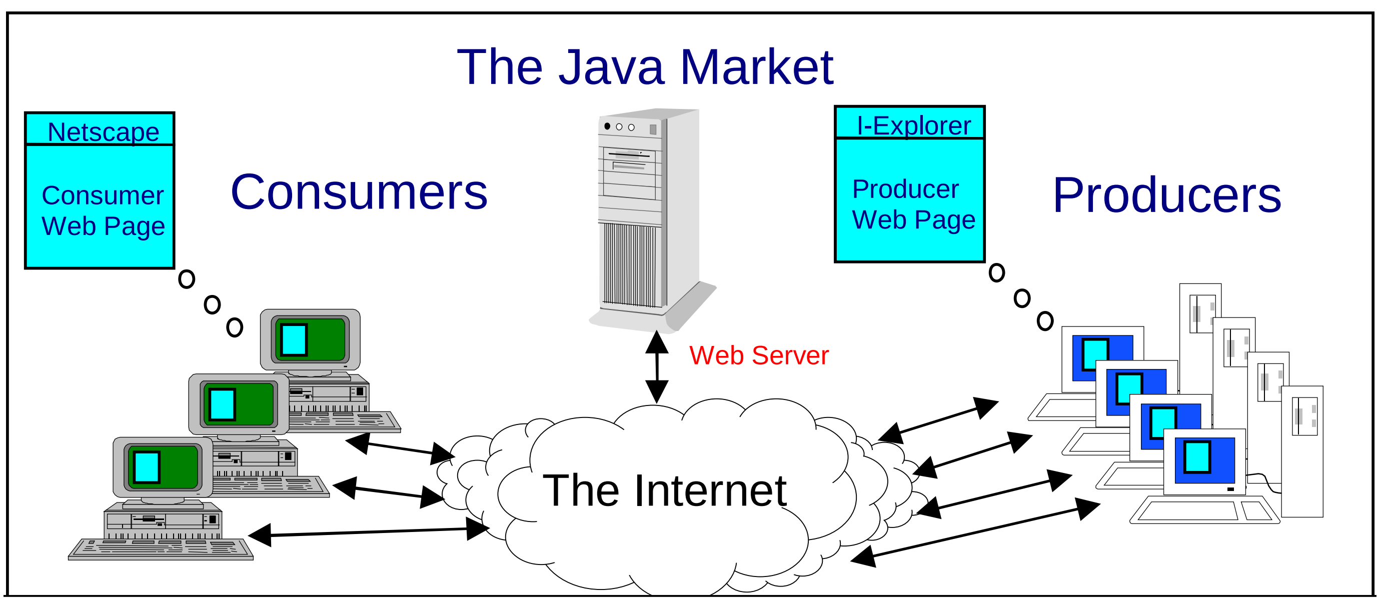


**Figure 5.2.1: An Internet-Wide Metacomputer**

## 5.3 Key Features

Producers and consumers can use the Java Market with minimal effort. The only programs that they need to run are secure Applets, which can be executed automatically by their Web browsers when they connect to the Market web pages. There can be no deleterious effects to the producer's machine.

As mentioned above, the Java Market is platform-independent. It can handle machines and task requests from anywhere on the Internet. A single Market can handle potentially hundreds of machines at a time -- almost all of the Market's work is done on the producer machines.

## 5.4 Implementation of the Cost-Benefit Framework

The Java Market uses the Cost-Benefit framework to determine what it is willing to pay for a resource and what it requires to accept a job. This built-in economic framework makes it easy to create limited Markets for exchanging resources between specific companies (using real money) or within a single organization (using virtual money). It can also be used to make an Internet-wide computational market.

The Java Market can convert the abstract cost functions associated with the marginal cost assignment strategy into real costs. This means that the Java Market's decisions are inherently comparable to those of the optimal off-line prescient Market.

## 5.4 Performance Measures

A metacomputing environment can be evaluated generally based on its performance in the following areas:

- Awareness of changing resource availability;
- Ability to handle resource heterogeneity;
- Guarantees regarding QoS (Quality of Service);
- Security;
- Scalability;
- Ability to impose a desirable scheduling policy;
- Transparency – must the user do extra work? and
- Speed.

Our two specific goals for the Java Market are:

- It should be able to apply the Cost-Benefit framework, and its associated algorithms, and
- It should be successful as a metacomputing system, able to use the power of the machines available to it to improve every user's performance.

How does the Java Market fare, regarding these measures?

Our system is continuously aware of all of its resources; when one of them disconnects or crashes, the Market will detect the loss of this resource and remove the relevant machine from its list of producers.

The Java Market can handle machines and task requests from anywhere on the Internet, and can impose its Cost-Benefit-based scheduling policy on them.

For these reasons, the Java Market is able to meet its first specific goal -- it can use the resource allocation algorithms associated with our framework.

The Java Market is optimized for security, at the expense of QoS guarantees. Rather than using the producer machines to the full extent possible, it operates within the Java "sandbox". This is a set of restrictions typically applied to Java applets that (in this case) protect the producer machines fully from hostile consumers. The Java Market also implements additional security to protect the integrity of the Market itself from hostile users.

The Market operates on the Java "virtual machine," which is an integral part of all Java-capable Web browsers. Such browsers are available on the vast majority of platforms.

The basic Market design does not allow transparency, but the Market has the next best thing -- ease of use. Using the Market is not the same as running a job on the local system, but it is a matter of a minute to upload a job or make one's machine available.

To measure speed, we performed 100 executions of a complex simulation, once distributing it via the Java Market and once doing it on a single machine. The Java Market was able to perform a 127-minute calculation in 35 minutes.

The Market's "resources" are machines throughout the Internet. It has the technical ability to manage these machines -- regardless of their architecture or location. The security of the Java sandbox makes offering one's machine to the Market in its off-hours a reasonable course of action. The Market has proven ability to harness this vast computational power; therefore, we believe it has met both of its specific goals.

# 6. ENHANCED MOSIX AND ENHANCED PVM

To demonstrate the generality of the Cost-Benefit framework, we present in this section a summary of an adaptation of the Cost-Benefit framework to a simpler metacomputing system: a network of workstations or a scalable computing cluster. This discussion is based on a joint research with the Mosix group [1, 2, 3, 4].

Enhanced Mosix and Enhanced PVM are two job assignment strategies for scalable computing clusters. These strategies also operate in a world defined by jobs and machines. However, we will assume that the machines are continuously available and benefit functions are non-existent – all jobs must be running on a machine from submission to completion. This research measures the performance of the marginal cost strategy in such an environment. The results indicate that it performs extremely well. (For a complete description, consult [1]).

## 6.1 The Model

Each job *j* submitted to the system is defined by the following properties:

- Its arrival time, *a(j)*, and
- Its resource vector, *r(j)*.

The resource vector *r(j)* still represents the various system resources the job requires to complete. The Cost-Benefit framework can handle any number of resources, but in our study we focused on two: normalized CPU seconds and memory used. We assumed that the memory requirement was known when a job arrived but that its duration was not.

On some systems, such as computers using the Mosix [2, 3] enhancements to the BSDI Unix-like kernel [11], jobs can be moved from one machine to another while running. On others, this is impossible. We will consider both cases.

Each machine *m* associated with the system is defined by its resource vector. Machines have no externally assigned cost to use and they are always available. The two resources considered for machines were memory size and CPU speed.

We assumed that machines were fair: if *k* jobs were running on a machine, each received 1/*k* of the CPU. If the machine was thrashing – that is, its memory was overutilized – all jobs were slowed down by a constant factor due to disk paging. Our performance measures were as follows:

- The **best time** for a job is the time it would take to run from start to finish while alone on a machine with the fastest CPU in the cluster and unlimited memory.
- The **slowdown** of a job is the ratio of the time a job actually takes to run and its best time.
- The **average slowdown** is the average over all jobs of their slowdowns.

## 6.2 Implementation of Our Framework

The Enhanced PVM strategy for job assignment is a direct extrapolation from the general marginal cost strategy. An exponential cost function is assigned to each resource, based on its utilization. Jobs are assigned to machines so as to minimize the total cost of all resources on all machines.

Processing power – the CPU resource – is, of course, not a resource that can be depleted. Fortunately, the general model can be extrapolated to handle this case without losing its theoretical guarantees. At any given time, the “maximum utilization” of the CPU resource is the highest load seen on any machine in the history of the system, rounded upwards to the nearest integral power of 2. The current utilization is the load on a given machine.

The Enhanced Mosix strategy assigns jobs in the same way, but can also move jobs from one machine to another to reduce the total cost of the system. The Mosix model of reassignment was used in our tests; in a given interval of time each machine could connect to a random set of other machines to exchange jobs with them.

## 6.3 Experimental Results

We created a Java simulation of a scalable computing cluster and examined four different methods of job assignment. One was the round-robin job assignment of the popular PVM system; one was the system used by Mosix; and two were the Enhanced methods described above.

Each execution returned the average slowdown over all jobs in that execution, as well as some information about the scenario itself. The results were evaluated in two different ways:

- The average slowdown over all executions is described in Table 6.3.1. The *average slowdown by execution* simply averaged the simulation results. The *average slowdown by job* weighted the simulation results by the number of jobs that arrived in that execution. The table describes 3000 executions.

- The behavior of Enhanced PVM and Enhanced Mosix improves in heavily loaded scenarios. This behavior is illustrated in Figures 6.3.1 and 6.3.2, detailing the first 1000 executions of this simulation.

| Slowdown for … | PVM | E. PVM | Mosix | E. Mosix |
|---|---|---|---|---|
| (average by execution) | 14.334 | 9.795 | 8.557 | 7.479 |
| (average by job) | 15.404 | 10.701 | 9.421 | 8.203 |

**Table 6.3.1: Average slowdowns in the Java simulation**

Experiments performed on a real system using real jobs, which necessarily run hundreds of times longer, indicate that the results obtained by the simulation are valid.

Each point in Figures 6.3.1 and 6.3.2 represents a single execution of the simulation. The X axis represents the slowdown using the unenhanced strategy, and the Y axis represents the slowdown using the Enhanced strategy. To

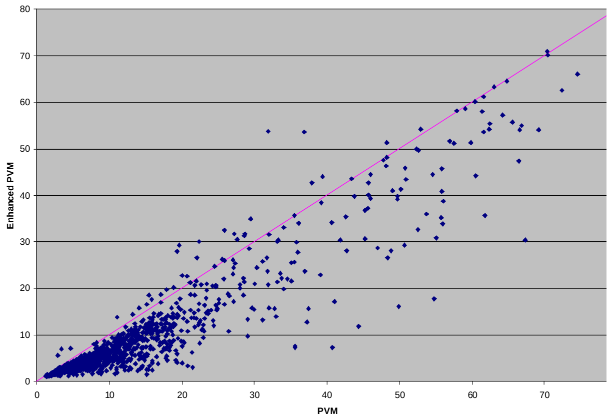


**Figure 6.3.1: PVM vs. Enhanced PVM**

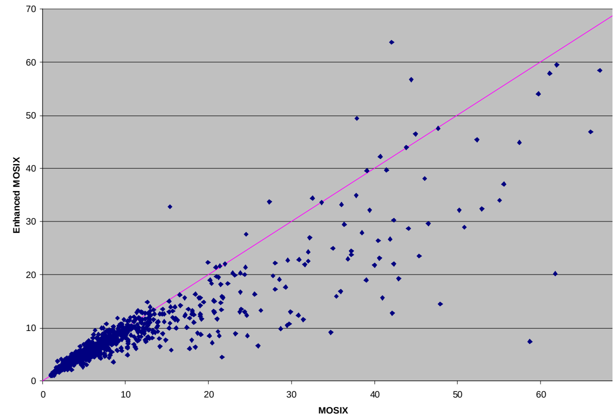


**Figure 6.3.2: Mosix vs. Enhanced Mosix**

the left of the 'x=y' line, the unenhanced strategy does better; to the right of the line, the Enhanced strategy does better. The graphs indicate that the "harder" a simulation was for all four strategies, the more likely it was that the Enhanced strategies would outperform the unenhanced strategies.

## 7. CONCLUSIONS

Any network of machines, strong or weak, can benefit from a wise resource allocation policy. Computational markets like the Java Market and powerful "metacomputing systems" like Mosix make it possible to borrow resources from unused machines. Even with that added power, however, a poor job assignment strategy can cripple the system. A computational market can lose real or virtual money by assigning important jobs to slow machines – which can easily happen if the faster machines are running less important jobs. A scalable computing cluster can have a heavily unbalanced load and thrashing machines if its resources are not well managed.

The Cost-Benefit framework is a universal system for efficient allocation of heterogeneous resources and jobs. This system is inspired by economic principles. Not only does it provide a simple conceptual framework to understand complex resource allocation issues, but it also proves itself valid in practice.

## 8. ACKNOWLEDGEMENTS

We would like to thank Amnon Barak, Arie Keren and Yossi Azar for their contribution to the Cost-Benefit framework. Amnon and Arie collaborated with us in the development of Enhanced PVM and Enhanced Mosix.

## References

[1] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, A. Keren. An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster. Available as tech report CNDS-98-2 at http://www.cnds.jhu.edu/publications.

[2] A. Barak, S. Guday and R. Wheeler. The Mosix distributed operating system, load balancing for Unix, Volume 672, May 1993.

[3] A. Barak and O. La'adan, The MOSIX Multicomputer Operating System for High Performance Cluster Computing, Journal of Future Generation Computer Systems, April 1998, to appear.

[4] The Mosix Multicomputer Operating System for Unix. http://www.cs.huji.ac.il/mosix/.

[5] The LYDIA Project (goal-oriented scheduling). http://www.ics.forth.gr/pleiades/projects/LYDIA/.

[6] C. Waldspurger. A distributed computational economy for utilizing idle resources. Master's thesis, MIT, Dept. of Electrical Engineering and Computer Science, May 1989.

[7] Condor. http://www.cs.wisc.edu/condor/.

[8] N. Camiel, S. London, N. Nisan, O. Regev. The Popcorn Project -- An Interim Report, Distributed Computation over the Internet in Java. Sixth International World Wide Web Conference, April 1997

[9] MILAN: http://www.cs.nyu.edu/milan/milan/index.html

[10] JAVELIN: http://www.cs.ucsb.edu/research/javelin/

[11] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin and O. Waarts. On-Line Machine Scheduling with Applications to Load Balancing and Virtual Circuit Routing. In *Proceedings of the ACM Symposium on Theory Of Computing (STOC)*, May 1993.

[12] Berkeley Software Design, Inc. http://www.bsdi.com