Aspen: Buy and Sell Spare CPU Cycles

Sam Davies, Martin Ouimet, Reuben Sterling
{sdavies, mouimet, benster} @mit.edu

***Abstract*— We describe a system that allows computer owners who have unused cycles to rent out their machines to people who want to run many computational jobs. The system is designed to run 1-2 hour jobs with deterministic results that do not require intermediate communication between nodes or computation checkpoints. 1) We introduce a payment and accounting scheme according to a simple measure of worker machine quality: the speed of computing jobs, 2) we implement policy to detect cheating workers, 3) we safely handle worker node failures through replication jobs across multiple machines, 4) we allow untrusted CPU owners to modify their worker client code, and 5) we check the correctness of clients by comparing replicated results.**

I. Introduction

Aspen allows people in need of extra computing resources to purchase computing time from other computer owners. Typical uses of computing resources include Pharmaceutical research [8], indexing algorithms [2], and machine learning [9]. Organizations involved in this type of research could benefit from a need-based computing grid [8].

Unlike other grid computing initiatives, Aspen enables any computer owner to make his or her computing resources available to anyone who needs extra resources. Computer owners receive financial compensation in exchange for completing computations requested by those in need of computing resources. Aspen creates a free market for the exchange of computer resources, where sellers and buyers set their own prices based on the quality of the resources they offer or require.

Although many examples of grid computing exist today, they are limited in scope. Projects like SETI@home [13] only benefit one organization and require charitable users to offer their spare resources for free. Projects like PlanetLab [12] make computation resources available to the general population, but offer no financial incentive to project participants. Furthermore, PlanetLab operates in a community model and requires users of resources to contribute to the resource pool by offering their own computation resources. Aspen is unique because it provides a financial incentive to users who contribute computation resources and allows any user with money to benefit from those available resources.

Aspen is intended to facilitate distributed processing of computations that are easy to parallelize. That is, Aspen works best with computations that can be broken down into small jobs whose results can be computed solely as a deterministic function of their arguments. For example, Aspen makes it easy for programmers to distribute to different computers computations performed inside the body of a loop. The types of computations that are suitable for Aspen require that each parallelizable unit of computation be deterministic and run long enough to justify the overhead of network communication.

Offering financial compensation to CPU owners increases the potential computational resources, but adds new challenges to the system; for example, there is a substantially higher motivation for sellers to try to cheat the system; Aspen detects cheaters by replicating jobs and comparing the results and the completion times of each job. Furthermore, Aspen has a responsibility to ensure that buyers of computing resources receive the resources they are paying for. For example, the purchased resources could potentially degrade over time; when the system slows down, whether it is due to network latency, load on the seller's machine, or intentional seller cheating, Aspen uses benchmark testing to detect the performance degradation and to take corrective action. In the event that a purposeful slowdown by the seller (to earn more money) is detected, Aspen will not charge the buyer of the resource and will not pay the seller.

This paper is divided into 11 sections. The Goals section identifies the key problems that Aspen addresses. The Design section explains the main functionality of the Brokerage System and Runtime Architecture and specifies how parties on either side can participate in the Aspen project. The Goals Revisited section explains how the presented design meets the stated goals. The Alternative Designs section lists a few of the alternate approaches that were considered and explains why the chosen approach is superior to the alternatives. The Relevant Work section situates the contribution of the

Aspen project in relation to existing grid computing projects. The Future Work and Conclusion sections summarize the contributions of the Aspen project.

A. Terminology

This paper uses the following terms:

Consumer: A machine that must execute a large suite of parallelizable jobs, has inadequate computation resources, and requires other machines to perform its computations in a timely manner. The consumer owner wants to execute the aggregate computations faster than his resources allow, and is therefore willing to pay money for extra computation resources.

Worker: A machine that owns excess computational resources and makes those resources available to consumers. The worker owner offers his machine's resources to the Aspen community for a specified minimum hourly price.

Job: A job is the smallest unit of computation on Aspen and runs on a single machine. A consumer's parallelizable computation will be broken down into many jobs. Jobs must be deterministic, and therefore may not make use of sources of randomness such as random number generators or multiple threads. We introduce this restriction because Aspen uses redundant computations to ensure correct results. A job is expected to run on the order of 1 or two hours. A Consumer is likely to run many jobs as part of a single large computation. A simple example of this is style of computation is the ”map” function [2] which applies a single function to each item in a list and produces a list of results. Aspen allows parts of the list to be sent to different machines so that the function may be run in parallel before the results are collated on the consumer's machine.

Aspen: In the context of the Aspen system design, the term Aspen refers to one or more servers that, under the control of the Aspen project, distribute jobs to workers, return results to consumers, manage executing jobs, and handle financial transactions.

Aspen System: The term Aspen system encompasses the entire computation network, including Aspen, all workers, and all consumers.

II. Goals

In light of the enumerated challenges, Aspen seeks to fulfill the following goals:

A. Make spare computing resources generally available, cost-effectively

Aspen should enable anyone with a fast and reliable Internet connection to rent out spare CPU cycles. Furthermore, it should allow anyone with a credit card or a Paypal account to use these resources. The payment to the Worker should cover at least the cost ofInternet connectivity and the cost of electricity while Aspen is using the system. The pool of resources Aspen offers should be attractive and affordable so that running computations through Aspen is a viable alternative to running computations locally. We include in this cost the extra effort necessary to modify existing code to use the Aspen system.

B. Operate with untrusted Workers

Because Aspen allows any Worker to join the Aspen system, Aspen does not have any control over a given Worker machine. As a result, Aspen cannot trust that all Workers will return the correct results of a computation. Furthermore, Aspen cannot trust that a Worker is executing a given computation at the agreed speed. Aspen should assume that the client software on some workers could be incorrect. Aspen should detect faulty or cheating workers and deal with them appropriately.

C. Operate reliably in a dynamic environment

Because Aspen does not have full control over a given Worker machine, Aspen cannot ensure the availability and reliability of a given Worker. It is quite likely that the available resources of a Worker will vary depending on a variety of factors (time of day, machine load, network state, etc.). Aspen should be able to operate reliably in the face of dynamic Worker behavior. The dynamic Worker behavior takes the form of failures and degraded resources. Aspen should gracefully handle these situations and still return the appropriate computation results to the Consumer.

D. Meet Consumers' pricing and timing constraints

Because Consumers will pay money to execute computations through Aspen, Aspen should ensure that Consumers get what they pay for. Consumers should be able to specify the speed of the machine they want to use and the maximum price that they are willing to pay. Aspen should ensure with high probability that Consumers are indeed given the appropriate resources and that Consumers are charged only for utilizing resources that meet their speed and cost specifications.

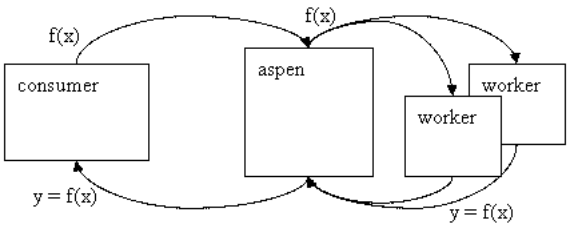

Fig. 1. The life-cycle of an Aspen job, computing f(x). 1. The consumer sends f and x to Aspen. 2. Aspen forwards copies to two workers. 3. The workers send back y, the result of computing f(x). 4. Aspen forwards y back to the consumer.

III. Design

A. Introduction to Aspen

We now describe our system, Aspen, which meets the above goals. The life-cycle of a job in the aspen system is as follows. 1. A consumer owner decides to run a computation job. She sends the binaries and the data to Aspen. 2. Aspen replicates the job on two or more worker machines which then perform the actual computational work. 3. Aspen receives the results from the workers. 4. Aspen returns the results back to the sender. From the Consumer's perspective, Aspen acts as a black box computing engine. It hides all of the details of worker tracking, cheating detection, and fault-tolerance from the consumer.

B. Nature of Computations suitable for Aspen

Aspen is tailored for particularly simple distributed computations that can be divided into a number of modular jobs. Each job must have the following qualities:

1) The time spent transmitting data and code should be small compared to the time spent performing the computation. This requirement limits the size of the combined consumer data and code. If the combined size is too large, the network bandwidth becomes the bottleneck, rather than the actual computation.
2) Jobs must be computable independently, without communication back to the consumer or any other worker. Worker code does not send communication over the network until the job is finished computing.
3) Jobs are deterministic. That is, the code defines a pure function from each argument to one correct result. We do not allow access to sources of randomness such as accesses to the system clock or multiple threads.
4) Jobs are relatively short. The longer a job takes, the more susceptible it is to network or worker failure. We therefore restrict maximum job lengths to around 1 or 2 hours.

C. Market

Aspen creates a free-market for the rental of computation. Workers advertise their speed and minimum hourly wage. Consumers set price and speed constraints that guide Aspen in selecting computers to run their jobs. Aspen selects Workers based on the Consumer's constraints and those required by Aspen's design goals. Constraints specified by the Consumer are maximum price and minimum speed.

D. Bundling Resources

Workers have many properties that may contribute to the speed of a computation. Some properties vary with time (such as CPU load and network usage), while others vary with hardware (such as memory, CPU architecture, cache size, etc.). Still other properties vary in their mean time to failure. Due to these variations, it is not always correct to say that one machine is faster than another–for example, certain computations may be sensitive to cache size, while other computations may require high network overhead. The Aspen project endeavors to bundle all of these variations into a single quality metric to evaluate and compare workers on a linear scale. We arrive at this evaluation by averaging the result of a diverse and representative set of benchmark jobs.

In the Aspen model, the basic unit purchased by a Consumer is an hour of computation on a single Worker. However, consumers need not purchase exactly one hour of resources; instead, when a Consumer sends a job to the Aspen, he agrees to pay a certain hourly wage, prorated depending on the job's completion time.

E. Market Value

The market value of a Worker is based on the computational speed of the Worker. The higher the computational speed of a computer, the higher a price its resources will fetch in the open market.

Aspen's design is built around the potential for untrusted workers, possibly running modified source code. Consequently Aspen cannot rely on the Worker to accurately report its own CPU speed or quality rating. To achieve good estimates on both these values, Aspen uses trusted external observations.

CPU speed is measured using benchmarking. When a user initially registers his machine to be an Aspen

Worker, Aspen sends the machine a series of unfamiliar benchmark jobs to test its performance. Aspen checks that the Worker returns correct values. Based on the latency of the benchmark jobs relative to the latency of the same jobs run on a local machine of known speed, Aspen estimates the effective CPU speed of the Worker. At this point, the CPU owner enters a contractual agreement to provide this same level of computational power to consumers. Over time, if Aspen detects that a Worker is offering resources that do not meet the contractual agreement (for example, due to a heavy local CPU load), Aspen reserves the right to revoke payment for a job.

During benchmarking, it is in the Worker owner's best interest to demonstrate his machine's true potential because of Aspen's competitive incentive scheme, which will be described in section III-F.

As a Worker continues to run jobs for the Aspen system, Aspen will periodically re-benchmark workers by replicating jobs on a local, trusted machine of known speed, and comparing the untrusted Worker's actual execution time with its expected execution time.

A variety of events may occur to cause an assigned job to fail: Worker crashing, network partitioning, or cancellation of a job by a Worker's owner. Aspen records all node failures that it observes. When advertising the quality of a machine to Consumers, Aspen takes into account the machine's mean time to failure. Failure-prone machines are likely to contribute to job slow-down just as slow machines are.

F. Worker Incentives

Aspen's goal of timely execution and accurate pricing requires that Workers execute code at the speed they have agreed upon. Aspen tries to guarantee that Workers abide by their agreement by paying Workers according to a competitive incentive system. As described in section III-A, all jobs run in Aspen are replicated on at least two machines, one of which may be a local, trusted, Aspen machine. The payment incentive system works as follows:

If the replicated job is run on an Aspen-owned machine:

1) The length of the job, with units in standard CPU cycles, is estimated by $L = t_b * s_b$, where t_b is the length of time in hours taken by the Aspen benchmarker machine to complete the job, and s_b is the estimated standard CPU speed of the Aspen machine in standard cycles per hour.
2) The effective CPU speed of the Worker is calculated by $s_w = L/t_w$, where L is the estimated length of the job calculated in step 1, and t_w is the time the Worker took to complete the same job.
3) The effective CPU speed of the Worker is compared against the Worker's advertised CPU speed. If the Worker's effective CPU speed is substantially slower (15%) than the speed recorded during benchmarking, the Worker does not get paid for completing that job.

If the replicated job is run on another Worker:

1) In this scenario, both machines are untrusted, so we must to choose one of them to consider as the "trusted" machine, against which we can make comparisons.
2) For both Workers, we calculate $L = t_w * s_w$.
3) We select the computer which yielded the smaller L (the lower amount of standard CPU cycles) as our trusted computer. The other computer is compared against the trusted one. If $L_{notrust}$ is within 15% of L_{trust}, then we pay both Workers. If not, only the "trusted" worker gets paid.

Using this comparison strategy, we provide an incentive for Workers to not overestimate their effective CPU speed. It is in the Worker's best interest to provide a realistic account of his resources. Complications that arise from this incentive system are addressed in section III-G.

G. Detecting Cheaters

Because there is real money at stake, and because we allow modified worker implementations, we must be prepared for the possibility that workers may try to cheat. A worker may try to cheat the system either by (1) speeding up jobs that it believes are benchmark jobs (thereby improving its published speed rating), or by (2) slowing down jobs that it believes are not benchmark jobs (thereby making Aspen think that the jobs took longer than they actually should have).

A worker may speed up its benchmark results by simply making up a return value and returning it as fast as possible. To prevent this style of cheating, we compare the deterministic results of jobs replicated on different machines. If the replicated results differ, something must be wrong. If this situation is encountered, Aspen will have to replicate the job on another external machine or on an internal trusted machine to determine the culprit. Once Aspen determines which worker is

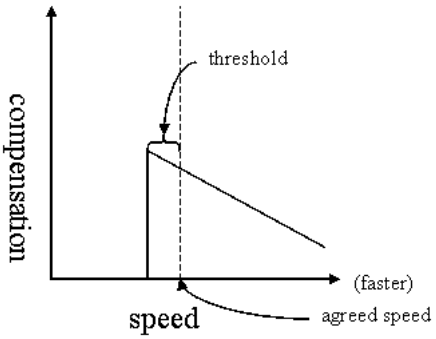

Fig. 2. Worker compensation vs. worker speed. If the worker runs jobs faster than its agreed speed dictates, the jobs will appear shorter and the Worker will be compensated less (because he performs less work). If a worker W1 runs a job significantly slower than its agreement specifies (according to a threshold set by Aspen), then another worker, W2, running a replication of the same job will complete sooner (with respect to its own speed). This allows Aspen to determine that W1 either has a loaded CPU or that he is cheating. W1 should therefore receive no compensation.

responsible, it records the suspicious behavior, and does not pay the Worker for the job.

Non-benchmark jobs take an unspecified amount of time to execute. Aspen cannot trust the consumer to tell us how long their job should take (they might have made a mistake while analyzing their code, or they might be trying to cheat). Furthermore, the halting problem states that it is provably impossible to predict the running time of an arbitrary program. Since Aspen cannot afford to run every job on its own trusted machines, Aspen must place some trust that the workers will execute jobs at the agreed speed. However, Aspen has a strategy to detect intentional or unintentional worker slow-down.

As seen in Figure 2, and described in III-F Aspen provides an incentive for workers to operate at the speed they have agreed to during benchmarking. For example, if a job takes 1 hour to complete on worker W1, a 1 GHz machine, and it takes 2 hours on W2, a 250 MHz machine, then Aspen knows that W1 ran half as fast as it could potentially have run (assuming, of course, that W2 was running at its agreed speed). It must therefore be the case that W1 had a loaded CPU or that the owner was intentionally cheating with an altered worker implementation. In either case, W1 will not be compensated for its work.

The speed of a worker is inherently variable due to many different factors: a worker's CPU load and network bandwidth fluctuates over time, instructions execute at different speeds on different CPU architectures, workers may have different amounts of pageable disk space, memory, and cache. We therefore introduce a threshold to account for this variability. The threshold defines the minimum acceptable speed of a worker in comparison to its agreed speed.

However, due to this threshold, it is possible for a worker to abuse the system if it knows the threshold, by slowing down just enough to still be acceptable (say 15%). If all registered workers behaved this way, it would be possible for the system to undergo a simultaneous slow-down creep. Aspen prevents this instability by randomly choosing some jobs to run on trusted Aspen-owned machines to determine the true job length. While some workers may get away with gaming the system for a short period of time, they do not know when we may choose to verify their behavior, and the odds are that they will eventually get caught.

H. Fault Tolerance

Aspen operates across an unreliable network with unreliable machines. Furthermore, the owners of worker machines may decide to temporarily disable the worker client code (for example, when they are using their machine for their own purposes). Aspen must therefore be tolerant of workers who come and go. To accomplish this, Aspen tracks the status of workers by pinging them periodically. If a worker becomes unreachable for a few minutes while running a job, Aspen assumes it has failed, starts a new replication of the job, and does not pay the failed worker.

Since workers may fail at any time, it is difficult to guarantee the running time of a job. Though Aspen does not know the expected running time of a job, the consumer may have expectations about the total running time. To try to keep jobs on time, Aspen must keep track of worker reliability. Ultimately, Aspen's perceived usefulness will be affected by the reliability of its workers and Aspen's ability to handle unreliable workers.

IV. Goals Revisited

A. Make spare computing resources generally available, cost-effectively

The purpose of this goal is to open Aspen to a large pool of potential workers, and to allow anybody to be a consumer. We can safely say that the current Aspen model enables this type of open community. Furthermore, the free-market guarantees that prices will settle in a fair range for both Consumers and Workers. In the ideal market system, Workers would behave in such a

way as to become as attractive as possible to potential Consumers. Consumers will likely be more interested in paying Workers more if they have a higher guarantee of expected performance on the Worker's part. Aspen tracks the historical behavior of each Worker and weeds out any Workers that do not comply with their agreed speed. It is not clear what the actual prices will be and we certainly anticipate that the prices will vary based on the type of Consumer and the type of computation he/she would like to perform. Nevertheless, the free market system will ensure that Workers will not be utilized at a rate under the minimum rate they specify. Furthermore, Aspen guarantees that Consumers will not pay for resources that do not meet their specifications. As both parties settle into an equilibrium that makes both of them happy, the resulting cost should be cost-effective with regards to each party's expectations.

B. Operate with untrusted Workers

Since any computer is eligible to be a worker, Aspen must be able to handle unexpected Worker behavior. The unexpected behavior can take the form of cheating by falsifying results and/or cheating by not adhering to the original performance agreement. Aspen is able to detect cheaters by replicating jobs on more than one Worker and by replicating jobs on a trusted Aspen machine. While complete cooperation between Workers might enable Workers to cheat, periodic replications on trusted Aspen machines will readily catch any cheaters. Aspen's competition-based incentive, however, makes it in everybody's best interest to play by the rules. The replication approach enables Aspen to assume nothing about new Workers, and it opens the possibility that a Worker can completely rewrite the Aspen Worker code. Furthermore, running benchmarks that are indistinguishable from actual computation jobs will enable Aspen even more enforcement capabilities. We certainly hope that Workers will understand that they cannot gain an advantage by cheating (quite the contrary) and that they can profit by playing by the rules. If the enforcement works well enough and cheaters are readily caught, honesty should prevail in the system, further encouraging fair play.

C. Operate reliably in a dynamic environment

Because Workers aren't trusted, Aspen must also be able to deal with unexpected behavior that is unrelated to cheating. Because Aspen does not control the Worker machines, Aspen has no guarantees about the network reliability and the "up time" reliability. Consequently, Aspen must deal with potential job failures due to Worker failures or network failures. With the constant pinging of Workers, Aspen keeps a clear picture of which Workers are up/down and which Workers are idle/busy. Aspen can easily detect that a Worker has gone down. The failure tolerance scheme enables Aspen to use replication and to recover from failed Workers. The failure tolerance happens transparently from the Consumer's point of view. Restarting jobs should not be too costly because of the nature of computations performed in Aspen (lost work will be at most 1 hour). In the case of restarting a job, a Worker crash is the responsibility of the Worker and hence the Consumer will not be charged for the running time before the failure occurred.

Another important source of dynamic behavior is the changing load on the Workers. Aspen can once again detect that a Worker is not meeting its advertised performance through replication. Aspen will also ensure that a Consumer is not being charged for computing resources that do not meet her specifications. Furthermore, by keeping a trust rating of Workers over time, Workers who constantly fail to meet their advertised performance and/or Workers who have demonstrated poor reliability will become quite unattractive. The free market system will drive those Workers down and eventually their usage will become quite rare.

D. Meet Consumers' pricing and timing constraints

Meeting the above three goals directly enables certain guarantees within Aspen. The final goal deals with guaranteeing certain properties of the computation to the Consumer. Because of the nature of the system, Aspen will never be able to give 100% guarantees about the performance or reliability of Workers. However, through replication, fault-tolerance, cheating detection, performance tracking, and compensation based on fair-play, Aspen can achieve a high level of predictability. The level of predictability will increase with the maturity of the system. Aspen can guarantee that Consumers will not pay for computations that have not completed and for workers that do not meet a Consumer's speed specifications. As the system matures and as Consumers become more sensitive to how long they expect their computations to take, we envision that Aspen will be able to fully guarantee that those expectations will be met.

V. USABILITY

For Aspen to succeed, it is crucial that the system be conceptually simple and easy to use for both the consumers and workers.

A. Consumer

The consumer must be able to parallelize her code in an intuitive and partially automated way. The following code samples provide an example of the manual source code modifications that are required to run code in Aspen.

Old Code:

```
Compute c = new Compute();
// Each loop of this 'for' block
// could be computed in parallel,
// but is currently serialized.
for(int i=1;i<xs.length;i++) {
  ys[i] = c.f(xs[i]);
}
```

Command line:

```
aspenize
  -source_class=Compute
  -source_function=f
  -target_class=Compute_aspen
  -target_function=aspen_f
```

New Code:

```
import Aspen.*;
...
Aspen.authenticate("username",
Aspen.passwordPrompt());
// Will pay 4 cents per hour
Aspen.setMaxPrice(0.04);
// Should be at least as fast as
// on a 1 GHz CPU
Aspen.setMinSpeed(1.0);
//instantiate the "aspenized" class
Compute_aspen c = new Compute_aspen();
// Method call replaces the entire
// for' loop in the original code
// Entire arrays are used as argument
// and return value.
ys = c.aspen_f(xs);
```

The old code sample simply loops over an array of arguments, applies a function to these arguments locally, and stores their results in another array. The command line produces a new class, Compute_aspen, that extends Compute and has a new method aspen_f(). This class automatically handles the details of sending new jobs to Aspen and collating the results as they return. In the new code, the consumer must authenticate herself to Aspen with a username and password. She must also specify a maximum acceptable price per CPU-hour and minimum acceptable execution speed. Aside from these changes, however, the consumer can leave the rest of the code completely unchanged.

1) Debugging: Despite developers' best testing efforts on their local machines, unexpected errors are bound to occur on remote machines. For example, code that runs smoothly on a Consumer machine might attempt operations that are restricted by the security policy of a Worker machine (like disk access). Aspen assists the software developer in correcting such errors by propagating any exception or error thrown during the execution of the code back to the Consumer.

B. Worker

In order for Aspen to succeed, it must be very easy for CPU owners to participate. For that reason, installation and setup of the aspen worker client program should be extremely simple, and the software should require little or no user interaction. At installation time, a CPU owner need only register their payment information and their minimum desired hourly wage. After installation, the client program will be automatically started each time the machine reboots. Jobs that meet the minimum hourly wage requirement will be started without user intervention on the Worker machine. Furthermore, the Aspen job will run as a low priority process with limited memory resources, so that it will not significantly hinder the regular availability of the CPU to its owner. Aspen tracks the history of jobs that were run on each Worker and keeps this information in persistent storage. Worker owners are given access to this information through the Aspen website.

VI. SECURITY

The notion of security in the Aspen system is very broad. Each type of participant in the Aspen system has different needs and expectations of what kinds of assurances the system should provide.

A. Worker Security

Any situation where a machine runs foreign code from untrusted sources creates critical security concerns. Unrestricted foreign code could create, delete or modify arbitrary files, generate unreasonable network traffic, or even install unwanted software. Even

non-malicious code might accidentally corrupt memory or files, hog processor cycles, or throw up dialog boxes. Our system must give Workers a high level of confidence that the arbitrary code they accept from Consumers cannot harm their system or inconvenience them in any significant way.

We handle such concerns in several ways. First, Consumer code is run at a low priority so that its execution cannot usurp the host machine's CPU resources. Second, the foreign code only has access to a limited memory address space. Next, we "sandbox" the foreign code so that it has very few system privileges. In our prototype implementation, foreign code running on a Worker is restricted to accessing and processing its allocated memory, and returning a result. The code may not access the disk, communicate across the network, or utilize the GUI. Thus, the Worker may be assured that foreign code may not cause damage to the machine or usurp its resources beyond what is explicitly allocated.

A Worker would like to be assured that he is properly compensated for any work he performs. Since the Worker must trust Aspen to properly pay the Worker for computation, the risk exists that a malicious third party might try to fool the Worker into doing illegitimate work without compensation. Aspen solves this problem by implementing an authentication and authorization certificate scheme between the Aspen servers and the Worker. Workers will only accept connections from an authenticated Aspen server and will only execute code that has been signed by Aspen.

A Worker should also be somewhat concerned about other machines claiming to be the Worker. Although a Worker would not care if another machine performed work for which he was being compensated, a malicious machine might want to cause the Worker to have a low rating by drawing jobs away from the Worker and purposely failing to return results. Because of such possibilities, the Worker may also wish to authenticate himself to Aspen before he can receive a job.

B. Consumer Security

The Consumer has significantly different security expectations than the Worker, since he is not executing foreign code or receiving compensation. Since a Consumer pays for each job he sends through the Aspen system, he would like to be assured that no other Consumer may spoof his identify and solicit computation time with his money. To prevent such spoofing, all jobs sent from a Consumer are qualified with a username and password, which the user sets up when he first registers with Aspen.

VII. Payment System

Aspen acts as a broker between private parties looking to exchange money for services. Financial transactions do not occur directly between the private parties, but rather between the private parties and Aspen. Aspen distributes income received from Consumers among Workers, thereby hiding the source of the money. Aspen takes a cut of all money traveling through its hands as its source of income.

A Consumer owners may provide Aspen with a credit card, a bank account, or a Paypal account from which Aspen can draw funds. Consumer owners may also set up a debit account with Aspen from which Aspen may deduct money as jobs are performed. Workers may provide Aspen with a bank account or Paypal account to which Aspen may deposit funds.

Since individual transactions are likely to involve very small sums of money, and because transactions of the nature described above involve per-transaction fees, Aspen waits for a significant balance to accrue on the Workers' account before paying them. We expect such transactions will usually occur in monthly intervals. Consumers may or may not be charged immediately after their jobs complete. The many transaction fees arising from many Consumer transactions may be offset by potential income from investing money for the period of time between collection from Consumers and distribution to Workers.

VIII. Alternative Designs

In designing and implementing Aspen, we considered a number of different design alternatives which we describe below.

A. SSH Model

An alternative way to provide distributed computation is via remote login. In this model, the consumer would be granted access to executing commands from a shell in real time (such as ssh or rlogin). If a computation fails, the consumer could simply restart it. While this model appears to have more flexibility, it provides no means of verification and measurability. Benchmarking will not work, since the worker may decide to give 100% of its CPU to a benchmark program, while it may decide to allocate only 1% of its CPU to the consumer without any repercussions.

B. Inter-Job Communication

If workers executing related jobs were allowed to communicate over the network, workers may be able to execute large computations more efficiently (such as distributed sorting). However, since network packets are routed in a non-deterministic order, the results may be non-deterministic. In general, it is difficult to ensure determinism of processes communicating over a network. Non-deterministic jobs prevent Aspen from detecting cheaters.

C. Per-Instruction Payment

Rather than timing computations, we may have considered counting the actual number of machine instructions the machine executed. To prevent cheating, we would compare the counts from the two machines. If they differed, we would decide that the higher count was cheating. However, since jobs may run on a wide variety of worker machines, this measure does not directly measure what we are paying for: the computation time necessary to run a job. For example, a multiply instruction may take 3 cycles on one processor and 5 on another. Furthermore, we would like to allow for dynamic optimizations of code ”hot spots” in software. These dynamic optimizations may alter the number of actual machine cycles used.

D. Long Jobs

Aspen relies on the assumption that jobs take a relatively short time to complete (on the order 1 or 2 hours). While this assumption is valid for a large class of problems, other classes of problems may require each job to run for a longer time. For long running jobs, the current fault tolerance mechanism would be inappropriate because the cost of lost work would be much more significant. If Aspen were to support long running jobs, it would need to introduce a checkpoint mechanism so that intermediate work steps can be recorded to minimize the cost of Worker failure before job completion. Putting a limit on the running time for jobs enables Aspen to keep Consumer implementation simple (no checkpoint required) and to make guarantees with a higher level of confidence (as to what performance a Consumer is getting from a given computer).

E. renting with more complicated quality statistics

One way that we may have built Aspen is to advertise each individual characteristic of worker machines separately: CPU speed, cache size, memory, hard disk space, network speed, etc. This would have given users more fine-grained control about what type of machine they are hiring to do computation for them. We chose instead to take the simple route, bundling all of these characteristics into a single quality measure reflecting the overall system’s performance for ”typical” jobs. This simplified quality measure is valid if most user tasks use resources in similar ways or if most worker machines offer similar resources. We are operating under the assumption that at least one of these conditions is true. Furthermore, it is much more difficult to design benchmarks to remotely verify the other system properties of a computer such as cache size, disk space, etc. For the time being, we believe that the simplest observable property–the job completion time–is sufficient for the level of accuracy desired by consumers.

F. scheduling jobs ahead of time

We may have considered a payment model that involved reserving machines ahead of time for particular time slots. This would allow for sophisticated scheduling algorithms to take into account user’s urgency preferences to prioritize jobs in times when Aspen is under heavy load. However, since Aspen has no way of predicting the true length of a job before it is run, we have chosen to allocate computing resources on demand in real time. In this way, machines may be used as soon as they become available. We believe that as the number of workers grows, there will always be some number of available workers that meet the consumer’s specifications.

IX. Relevant Work

Internet-wide grid computing has been around since the mid 1990s. The pioneer projects of Internet-wide grid computing are the SETI@Home project [13] and the Distributed.net project [3]. More recent projects of the same flavor include the Folding@Home project [5]. These projects are special-purpose applications that require computer owners to go to the relevant project web site and download the specific binaries for each project. The motivation for computer owners to join the grids of each project relies largely on a sense of goodwill and a willingness to participate in the advancement of a valuable cause (e.g. genetic code sequencing through protein folding [5]). None of these projects involve financial contributions for its participants. Furthermore, these projects are specific to a single application and have no aspiration to enable generic grid computing.

Conversely, various organizations have sought to provide the general-purpose infrastructure that is re-

quired for generic Internet-wide grid computing [1]. Among these organizations, UNICORE [14], Gridbus [7], Globus [6], and Legion [10] appear to dominate the literature. The focus of these organizations has been on defining the enabling technologies and protocols. However, these organizations do not provide a brokerage service to mediate the relationship between computer owners and organizations. Furthermore, none of these infrastructures mention incentives for owners to offer their computers or for commercial organizations to utilize the computational grid.

Furthermore, other projects aim to create large node communities that can be utilized for general purpose computations. Such projects include PlanetLab [12] and Emulab [4]. In the case of Emulab, the grid is completely owned and managed by Emulab. In the case of PlanetLab, the grid contains arbitrary nodes around the Internet. However, both of these projects are available for non-commercial research purposes only. Moreover, these projects rely heavily on the goodwill of participants in matters of security, resource utilization, and reliability.

Internet-wide grid computing is becoming mature technologically. The remaining piece of the puzzle lies in popularizing grid participation both for commercial organizations and for computer owners. Existing projects such as the SETI@Home project have been successful by relying mostly on the goodwill of participants. The Aspen project seeks to unleash the full power of grid computing by providing financial incentives to grid participants. Furthermore, the Aspen project seeks to popularize grid computations among commercial organizations by supplying an infrastructure where organizations can execute computations reliably and safely in a time and cost-effective manner. A survey of the literature has shown that these issues have yet to be addressed.

X. Future Work

The Aspen project team plans to pursue several avenues to bring this product to fruition in the coming months. We have already communicated with the Microsoft iCampus awards team, as well as with the Athena I/S staff about the possibility of installing a large prototype testbed at MIT. We also hope to begin deploying the final version of the software with the help of commercial software developers.

One of the team members is already using the Aspen system for a computer vision project.

XI. Conclusions

In this paper, we proposed a system that addresses the practical issues of scaling distributed computation to the global, internet-wide scale. We address issues of measurement and markets for exchanging computation for real money. We cover issues of cheating that may arise from workers who are free to modify the code that resides on their own machine, and we tolerate node and network failures.

The Aspen team believes that this system has the potential to be a practical solution to people who are in need of more computing resources. We therefore plan to follow through with the next stages of development and deployment.

XII. References

[1] Asadzadeh, P., Buyya, R., Kei, C. L., Nayar, D. Venugopal, S. Global Grids and Software Toolkits: A Study or Four Grid Middleware Technologies. Grid Computing and Distributed Systems (GRIDS) Laboratory. The University of Melbourne, Australia, 2004.

[2] Dean, J., Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. Google, Inc. To appear in Proceedings of the 6th Symposium on Operating System Design and Implementation OSDI 2004, April 2005.

[3] Distributed.net. http://www.distributed.net/.

[4] Emulab. http://www.emulab.net.

[5] Folding@Home. http://folding.stanford.edu/

[6] The Globus Alliance. http://www.globus.org/.

[7] The Gridbus Project. http://www.gridbus.org/.

[8] Ricadela, A. Slow Going on the global grid. Information Week. February 21st, 2005.

[9] Isbell Jr., C. L., Husbands, P. The Parallel Problems Server: an Interactive Tool for Large Scale Machine Learning.

[10] The Legion Project. http://legion.virginia.edu/.

[11] The Mathworks. MATLAB Distributed Computing Toolbox User's Guide. Version 1. November 2004.

[12] PlanetLab. http://www.planet-lab.org.

[13] SETI@Home. http://setiathome.ssl.berkeley.edu/.

[14] The UNICORE Grid System. http://www.unicore.org/.