Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments

Eric M. Heien, Noriyuki Fujimoto, Kenichi Hagihara
Graduate School of Information Science and Technology, Osaka University
Toyonaka, Osaka 560-8531, Japan
{e-heien, fujimoto, hagihara}@ist.osaka-u.ac.jp

Abstract

Internet based volunteer computing projects such as SETI@home are currently restricted to performing coarse grained, embarrassingly parallel tasks. This is partly due to the "pull" nature of task distribution in volunteer computing environments, where workers request tasks from the master rather than the master assigning tasks to arbitrary workers. In this paper we develop algorithms for computing batches of medium grained tasks with soft deadlines in pull-style volunteer computing environments. Using assumptions about worker availability intervals based on previous studies, we develop models of unreliable workers in volunteer computing environments. These models are used to develop algorithms for task distribution in volunteer computing systems with a high probability of meeting batch deadlines. We develop algorithms for perfectly reliable workers, computation-reliable workers and unreliable workers. The effectiveness of the algorithms is demonstrated by using traces from actual execution environments.

1 Introduction

In recent years, public-resource computing or "volunteer computing" projects have demonstrated the power of performing distributed computation using donated resources over the Internet. Projects such as SETI@home [3] and Folding@home [11] sustain computation speeds of tens or hundreds of teraflops, comparable with high end supercomputers. In these projects, computational tasks are distributed and executed on donated computers around the world.

Volunteer computing (VC) systems use a master-worker style of computing, where tasks are distributed from a master machine to worker machines and executed. Because such systems are composed of donated resources, they can make few guarantees about network or machine reliability. Therefore volunteer computing is usually applied to coarse grained embarrassingly parallel computation with tasks that require hours or days to complete. Also, because of the volatile nature of the donated resources, task completion deadlines are generally on the order of days or months.

VC environments differ from traditional grid computing environments in several important ways. First, because the worker machines in VC systems are owned by private individuals, communication and computation reliability is significantly lower. A worker machine may often be disconnected from the network, or used for other purposes without advanced warning. Second, worker machines are often behind firewalls and NATs which allow only worker to master connections. This means that a "pull" model of task distribution must be used, instead of the common "push" model where the master distributes tasks to arbitrary workers.

In this paper, we propose algorithms for meeting batch deadlines in VC systems given varying degrees of worker reliability. Rather than normal VC deadlines of days or months, we deal with deadlines of minutes or hours. We call this "low latency computing." For such pull-style task distribution, the key to meeting batch deadlines is ensuring that all tasks are distributed to workers in a timely manner and that workers complete the tasks before the deadline. To develop suitable algorithms, we first define the environment and analyze the effect of communication and computation unreliability in Section 2. Given worker availability distributions from previous studies, we show how worker task requests can be modeled as a Poisson process and task computation time can be modeled as a probabilistic distribution. Using these models, we develop algorithms for task distribution in Section 3. The algorithms are verified using trace-driven simulations in Section 4. Finally, we review related work and offer our conclusions in Sections 5 and 6.

2 Computation Model

In order to develop algorithms for task distribution in a volunteer computing environment, we first define workers,

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

batches and tasks in Section 2.1. In Section 2.2 we examine communication reliability in workers and use simulations to demonstrate that connections from semi-reliable workers can be modeled using a Poisson process. In Section 2.3 we examine computation reliability in workers, and derive a probabilistic function for deadline satisfaction based on worker speed and deadline time.

2.1 Computation Environment

In this model for VC low latency batch computing, there are M batches of work, denoted $B_0, \ldots, B_{M-1}$. Each batch B_i has N unit-length independent tasks denoted $T_0^i, \ldots, T_{N-1}^i$, a submission time S_i and a soft deadline D_i with $S_i < D_i$. All tasks in batch B_i are available for distribution at time S_i. Because workers are not fully reliable, tasks may not finish before the deadline. Our model puts a probabilistic bound on deadline satisfaction, therefore we use a soft rather than a hard deadline. Batches are sequential and do not overlap, i.e. $\forall i, n, (i < n) \Rightarrow D_i < S_n$. All tasks are initially on a single master machine. The goal of the server is to assign tasks to workers such that the number of tasks completed after their deadlines is minimized.

To compute the tasks, there are P workers $W_0, \ldots, W_{P-1}$. At any given time, a worker is in one of two states - available or unavailable. The master is always available. A worker in the available state may perform computation or communication, an unavailable worker may do neither. Various factors may cause transition between these states - user activity/idleness, machine reboot/shutdown, etc. If a worker transitions from available to unavailable while executing a task, the task is resumed at the same point when the worker returns to the available state. This behavior can be achieved through task checkpointing. Each worker W_i has a task computation time R_i, which is the number of seconds the worker requires in the available state to complete a task. The reconnection period T specifies the rate of task requests from a worker.

In this paper, worker unavailability intervals are assumed to be finite in length. In other words, no workers quit the computation forever. For actual systems this is not a correct assumption due to part failures, volunteers ending their participation, etc.. Analysis of volunteer computing projects indicates worker lifetime roughly follows an exponential distribution with a mean of 3 months [4]. In this paper we consider low latency batches with deadlines on the order of minutes or hours. Therefore, the probability of a given task failing due to a worker quitting is much less than 1% and we feel it can be ignored without significant loss of accuracy.

In terms of communication and computation, workers can be either reliable or semi-reliable. A worker is "communication-reliable" if it can guarantee communication with the master at an arbitrary time C. A worker is "semi-communication-reliable" if the probability of communication in time $[C, C+\delta]$ exponentially increases with respect to δ. A worker W_i is "computation-reliable" if it can guarantee task completion within a specified time limit R_i, which is the task computation time of the worker. A worker is "semi-computation-reliable" if the probability of task completion in time $[R_i, R_i+\epsilon]$ exponentially increases with respect to ϵ.

Table 1. Task request simulation parameters.

Parameter	Values
Number of workers (P)	500
Length of simulation	3 days
Reconnection period (T)	50, 100, 500, 1000, 2000, 5000 secs
Availability distribution	Weibull (α, β)
Weibull shape param. (α)	0.5
Mean availability (2β)	1000, 10000, 30000 secs
Unavailability distribution	Exponential (λ)
Mean unavailability ($1/\lambda$)	100, 1000, 5000 secs

2.2 Semi-Comm-Reliable Workers

Next, we examine the effect of worker unreliability on task requests and propose a model for task requests from VC workers. To develop a model for semi-communication-reliable workers, we make assumptions about worker availability and unavailability intervals, then perform simulations using these assumptions. The results of the simulations indicate that task requests from semi-communication-reliable workers can be modeled as a Poisson process.

Previous studies show that worker availability interval lengths can be modeled with a Weibull distribution [13]. We know of no detailed studies of worker unavailability intervals, but based on the results in [9] and our own analysis, we chose to model short term unavailability interval lengths (hours or less) as an exponential distribution. Long term worker unavailability intervals are more erratic due to business hours, holidays, etc. This is handled separately by tracking active workers, as described in Section 3.2.

Given the distributions of worker availability and unavailability, we performed simulations to determine a model for worker task requests. In the simulations, each worker transitions back and forth between available and unavailable, with the time in each state determined by sampling the above mentioned distributions. Each worker W_j connects to the master at an exponentially distributed initial time $C_j = Exp(\lambda = 1/T)$. After each connection, the worker is assigned a new connection time $C_j = CurTime() + T$. If a worker is unavailable at time C_j, the connection is initiated when the worker becomes available. Worker connection times were recorded, and we analyzed the distribution

Table 2. Mean interconnection gap lengths (seconds).

Mean Availability		1000			10000		30000	
Mean Unavailability		100	1000	5000	1000	5000	1000	5000
T	T/P							
50	0.1	0.1129	0.2062	0.6191	0.1185	0.1748	0.1135	0.1434
100	0.2	0.2171	0.3915	1.174	0.2305	0.3400	0.2210	0.2793
500	1.0	1.024	1.606	4.607	1.105	1.618	1.064	1.343
1000	2.0	2.027	2.813	7.378	2.160	3.119	2.097	2.622
2000	4.0	4.041	4.967	10.97	4.219	5.875	4.140	5.065
5000	10.0	10.13	11.12	18.20	10.34	13.11	10.25	11.88

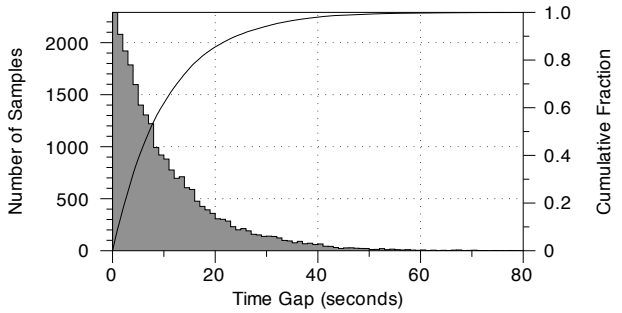


Figure 1. Time Gaps Between Connections ($T = 5000, \beta = 5000, \lambda = 10^{-3}$)

of time intervals between successive connections. The parameters for the simulations are shown in Table 1.

The results of one simulation are shown in Figure 1 with the time gaps between consecutive connections to the master plotted as a histogram and cumulative fraction. All results showed a similar pattern. Table 2 shows the mean interconnection gaps for given availability and unavailability intervals, compared with the expected interconnection gap T/P. As expected, higher levels of worker unavailability give a higher mean interconnection gap. This shows that for short unavailability intervals, $\lambda = T/P$ is a good approximation of mean interconnection time.

Hypothesis 1. *Let T be a positive time period. Let P workers with Weibull availability intervals and exponential unavailability intervals be assigned connection times as described previously. After each worker connects, it is assigned a connection time at T seconds in the future. Then the time gap between connections can be modeled as an exponential distribution function (EDF).*

To confirm Hypothesis 1, we applied the Anderson-Darling test (AD) [5] to all simulation results. We chose this test instead of the Kolmogorov-Smirnov test (KS) because of its higher sensitivity [15]. Tests were conducted to compare the simulation results with EDFs using different λ parameters - one based on the expected time between connections $\lambda = P/T$ and the one on the mean value of the actual results. The AD test is sensitive to imperfections in large data sets, so 1000 random samplings of 100 data points each were taken from each data set. The AD p-value for each of the samplings was calculated, and an average p-value was determined. Generally, the minimum acceptable p-value for the AD test is 0.05, so p-values higher than this indicate good fit with the EDF.

When comparing the samples with an EDF using the expected mean, average p-values ranged from 0 to 0.284. Simulations with high worker unavailability tended to have low p-values. However, when comparing the samplings to an EDF with mean equivalent to the data set, average p-values ranged between 0.205 to 0.296. A process with time between events following an EDF is defined as a Poisson process. Based on these results we offer Hypothesis 2.

Hypothesis 2. *Let T be a positive time period. Then connections from P semi-communication-reliable workers can be modeled as a Poisson process. For workers with high availability, the process rate parameter is $\lambda \approx P/T$.*

2.3 Semi-computation-reliable Workers

Next we propose a model for worker reliability in regards to computation speed and task deadlines. This model is based on the same assumptions of worker availability and unavailability interval distributions as the previous section.

We assume that the ratio of availability to unavailability is high, and tasks are sufficiently short such that a worker experiences either zero or one unavailability intervals while performing a task. Suppose a task is distributed to worker W_i at an arbitrary time C during the availability interval. The worker then requires R_i seconds in the available state to complete it. If the availability interval length is Weibull distributed, the probability of the worker completing the task before entering the unavailable state is:

$$\gamma = Pr(Weibull(\alpha, \beta)/2 > R_i) = e^{-(2R_i/\beta)^\alpha} \quad (1)$$

Then the total task completion time X is given by:

$$X = R_i + (1 - \gamma)Exp(\lambda) \quad (2)$$

where $Exp(\lambda)$ represents the unavailability interval length and is a random exp. distributed value with mean $1/\lambda$. The probability of this time being less than the deadline D is:

$$J(D, R_i) = 1 - e^{-\lambda \frac{D-R_i}{1-\gamma}} \quad (3)$$

3 Task Distribution Algorithms

In order to meet batch deadlines, tasks must be distributed to workers in a timely manner. Pull-style volunteer computing task requests go from workers to the master. Thus, a sufficient number of task requests must occur between the batch submission and deadline. In a VC system, this is done by requesting workers to connect at certain times. This is known as the "reconnection time" and is denoted C_j for worker W_j.

In this section, we describe algorithms for ensuring a high probability of sufficient task requests to complete all batches before their deadlines. Section 3.1 presents an algorithm for fully reliable (communication-reliable and computation-reliable) workers, and proves that it satisfies all deadlines in certain conditions. In Section 3.2 we provide an algorithm for semi-communication-reliable workers with a probabilistic bound on failure. The algorithm for semi-reliable (semi-communication-reliable and semi-computation-reliable) workers is given in Section 3.3, and also provides a probabilistic bound on failure. The effectiveness of these algorithms is demonstrated in Section 4.

3.1 Fully Reliable Homogeneous Workers

In this section we consider fully reliable workers, with guaranteed communication and computation times. Workers are considered computationally homogeneous and reliable, meaning that a task always takes R seconds and finishes at time $C_j + R$. To meet the deadline D_i, all tasks in B_i must be distributed to workers before time $L_i = D_i - R$. We assume that all workers connect before S_0.

In this algorithm the master accepts worker connections, sends an undistributed task to the worker from the current batch and notifies the worker of their next connection time C_j. Worker W_j connects at time C_j or immediately if the current time is past C_j. Upon connection the worker receives a task and next connection time, then executes the task. The algorithm for the master is shown in Algorithm 1. The workers are fully reliable, so the algorithm only needs to ensure that N task requests occur during each batch.

Given this algorithm and constraints on the task length, deadline/submission times, and number of tasks/workers, Theorem 1 proves that all batches will meet their deadlines.

Theorem 1. *Algorithm 1 results in all batches meeting their execution deadlines if and only if* $\forall i, D_i \geq S_i + R\lceil \frac{N}{P} \rceil$.

Algorithm 1 Fully Reliable Homogeneous Workers

```
1: AssignBatch ← 0, AssignTask ← 0
2: SendBatch ← 0, SendTask ← 0
3: while SendBatch < M do
4:    Get connection from W_j
5:    if CurrentTime() ≥ S_SendBatch then
6:       Send task T^SendBatch_SendTask to W_j
7:       SendTask ← SendTask + 1
8:    end if
9:    if SendTask ≥ N then
10:      SendTask ← 0, SendBatch ← SendBatch + 1
11:   end if
12:   if AssignBatch < M then
13:      C_j ← S_AssignBatch + AssignTask * (L_AssignBatch −
         S_AssignBatch)/(N − 1)
14:      Send reconnection time C_j to W_j
15:      AssignTask ← AssignTask + 1
16:   end if
17:   if AssignTask ≥ N then
18:      AssignTask ← 0
19:      AssignBatch ← AssignBatch + 1
20:   end if
21: end while
```

Proof. Proof by induction. At time S_0, by definition no workers are executing a task and all P workers have connection times. Next, assume that no workers are executing a task and all workers have been given connection times at time S_i. We now demonstrate that if $D_i \geq S_i + R\lceil \frac{N}{P} \rceil$, then all tasks in B_i will be finished at or before D_i and at time S_{i+1} there will be no workers executing tasks.

If $P \geq N$ then at time S_i all tasks for batch B_i have been assigned, and each worker will receive at most 1 task from the batch. The latest task distribution time will be $S_i + (N-1)\frac{L_i - S_i}{N-1} = L_i = D_i - R$ and the latest task completion time will be D_i, meaning that no workers are executing a task after D_i. If $D_i < S_i + R$, then $L_i < S_i$, which is a contradiction because no tasks from B_i can be distributed before S_i.

If $P < N$ then at time S_i only P tasks from batch B_i have been assigned. During the execution of batch B_i, a worker will request and execute either $\lfloor \frac{N}{P} \rfloor$ or $\lceil \frac{N}{P} \rceil$ tasks. Because a worker executes tasks one by one, the latest a worker will request a task is at time $max(L_i, R(\lceil \frac{N}{P} \rceil - 1))$ and the latest task completion time will be $max(D_i, R\lceil \frac{N}{P} \rceil)$. If $D_i < S_i + R\lceil \frac{N}{P} \rceil$, the batch completion time will be $R\lceil \frac{N}{P} \rceil > D_i$ and the deadline will not be met. In this case, there can be no guarantees about the execution state of workers at time S_{i+1}. If $D_i \geq S_i + R\lceil \frac{N}{P} \rceil$, the batch completion time will be D_i and no workers will be executing tasks at S_{i+1}.

Therefore all batches will meet their execution deadlines if and only if $\forall i, D_i \geq S_i + R\lceil \frac{N}{P} \rceil$. □

3.2 Semi-Comm-Reliable Workers

Next we consider homogeneous workers that are computation-reliable and semi-communication-reliable. In other words, workers with guaranteed computation but probabilistically bounded communication times.

In VC environments, predicting the future availability state of a given worker at a specific time is nearly impossible. Algorithm 1 cannot be used in such environments because it depends on each worker W_j being available at C_j. Note that Algorithm 1 does not request all workers to connect at time S_i, but instead spreads out the connections between S_i and L_i to maintain a constant rate of task requests. In the same way, semi-communication-reliable workers can meet a deadline by maintaining a stream of task requests.

In Section 2.2, we demonstrated that task requests from semi-communication-reliable workers can be modeled as a Poisson process. Given this model, we now determine how to calculate the connection period T so as to distribute all tasks before the batch deadline. Let V be the event where at least N task requests are sent to the master from P active workers in a given time period L. Given Hypothesis 2, we can control the probability K of V occurring by specifying a reconnection time C based on P, L and N.

The number of task requests occurring in a Poisson process follows the Poisson distribution, which gives the probability of exactly N task requests occurring in a given time period. Because this is a probabilistic model we can only put a bound on the probability K of a specified number of task requests occurring. For a probability K of at least N task requests in a given time period L, λ must satisfy:

$$K \geq 1 - \sum_{i=0}^{N-1} \frac{e^{-\lambda}\lambda^i}{i!} \tag{4}$$

Given λ from this equation, the reconnection period is:

$$T = \frac{PL}{\lambda} \tag{5}$$

The number P of active workers can change over long time periods. Therefore the value of T must change during the course of the computation. One way to track active workers is to count the number of workers which connected in the last T seconds. Algorithm 2 demonstrates how to use the active worker count to distribute tasks to semi-communication-reliable workers. This algorithm ensures sufficient task requests to the master before the distribution deadline. We demonstrate the effectiveness of this algorithm in Section 4.

Algorithm 2 Semi-Communication-Reliable Homogeneous Workers

```
Calculate λ from K and N; estimate P; T ← PL/λ
SendBatch ← 0, SendTask ← 0
while SendBatch < M do
    Get connection from W_j
    if CurrentTime() ≥ S_SendBatch then
        Send task T^SendBatch_SendTask to W_j
        SendTask ← SendTask + 1
    end if
    C_j ← CurrentTime() + T; Send C_j to W_j
    if SendTask ≥ N then
        SendTask ← 0, SendBatch ← SendBatch + 1
        P' = num workers in last T seconds, T ← P'L/λ
    end if
end while
```

3.3 Semi-reliable Workers

Finally, we propose an algorithm to replicate and distribute tasks to semi-reliable heterogeneous workers. These are semi-communication-reliable and semi-computation-reliable workers which follow the models in Sections 2.2 and 2.3. As demonstrated, the probability of tasks meeting a deadline is a function of worker task completion time R, deadline time D and availability/unavailability intervals. If R is a significant fraction of D or unavailability intervals are relatively long, the probability of meeting the task deadline may be arbitrarily low. Therefore we use task replication to ensure high probabilities of meeting task deadlines.

In Section 2.3 we derived a model for task completion rates in VC systems. This model was specified in terms of $J(D, R)$, the probability of a distributed task meeting the deadline. Similar to other stochastic scheduling research [14], we assume the task computation time can be modeled as a random variable. Because we can't know which workers will connect at which times, we use the mean task completion time - denoted $\overline{R}$ - to calculate J. The mean task completion time is the mean time among all workers to complete a unit task when computing at full speed.

If a task is replicated and distributed H times, then the probability of at least one task meeting the deadline is $Q = 1 - (1 - J(D, \overline{R}))^H$. Given N tasks, the probability of all tasks meeting the deadline is $E = Q^N$. For a probability of success E the appropriate number of replicas H is:

$$H(D, \overline{R}) = \lceil \frac{ln(1 - \sqrt[N]{E})}{ln(1 - J(D, \overline{R}))} \rceil \tag{6}$$

Although Equation 6 gives a high number of task replicas, this is only an upper bound. With an intelligent scheduling algorithm, the number of necessary replicas can be reduced through various methods. These include - not sending a replica for a task that has already been completed, favoring replication for tasks on slow machines and favoring replication for tasks distributed closer to the deadline.

Algorithm 3 shows the master task distribution algorithm for semi-reliable heterogeneous workers. This algorithm is

similar to Algorithm 2, except that we create replicas of the tasks and assign reconnection times such that all replicas are distributed with high probability. Because Algorithm 3 is based on the above model, it ignores whether a task has been completed when deciding to make a replica.

Algorithm 3 Semi-Reliable Heterogeneous Workers

1: Calculate λ from K, N, H; estimate P; $T \leftarrow \frac{PL}{\lambda}$
2: $SendBatch \leftarrow 0, SendTask \leftarrow 0, ReplTask \leftarrow 0$
3: **while** $SendBatch < M$ **do**
4: Get connection from W_j
5: **if** $CurrentTime() \geq S_{SendBatch}$ **then**
6: Send task $T_{SendTask}^{SendBatch}$ to W_j
7: $SendTask \leftarrow SendTask + 1$
8: **end if**
9: $C_j \leftarrow CurrentTime() + T$; Send C_j to W_j
10: **if** $SendTask \geq N$ **then**
11: $ReplTask \leftarrow ReplTask + 1, SendTask \leftarrow 0$
12: **end if**
13: **if** $ReplTask \geq H$ **then**
14: $ReplTask \leftarrow 0, SendBatch \leftarrow SendBatch + 1$
15: P' = num workers in last T seconds; $T \leftarrow \frac{P'L}{\lambda}$
16: **end if**
17: **end while**

4 Experiments

We conducted a series of experiments to test the algorithms in Sections 3.2 and 3.3. Simulations were implemented with the SimGrid distributed application simulator [12]. For these experiments, we used execution availability traces taken from several sources [9]. These availability traces were taken from a computer department cluster (LRI - about 40 workers), desktop PCs in an undergraduate classroom (DEUG - about 40 workers) and desktop PCs at the San Diego Supercomputing Center (SDSC - about 200 workers). There are currently no trace sets from VC systems, though the XtremLab project [1] is working to obtain such traces. To approximate a VC environment, we combined the LRI, DEUG and SDSC trace sets.

4.1 Semi-Comm-Reliable Workers

To confirm that Algorithm 2 provides sufficient task requests from workers, we performed experiments using the trace data described above. Although this trace data is not from semi-communication-reliable workers, the main purpose here is to demonstrate the efficacy of Algorithm 2 rather than simulate a realistic environment. The parameters for the experiments are shown in Table 3. These parameters were selected to represent a range of possible low latency applications, from large batches with a long deadline (Experiment 2) to small batches with short deadline (Experiment 3). The experiment results are shown in Figure 2.

Table 3. Task request simulation parameters.

Parameter	Experiment Number			
	1	2	3	4
Number of Batches (M)	32	16	64	64
Tasks per Batch (N)	64	128	64	128
λ derived from K, N	93	167	93	167
Batch Deadline (D)	1024	2048	512	1024
Batch Submission (S_i)	$S_0 = 0; S_{i+1} = S_i + D$			
Task Completion Time (R)	256 seconds			
Target Success Probability (K)	0.999			

The figures show the active worker count at the start of each batch, and the number of task requests received during each batch. The dark histogram represents the number of task requests that arrived before the batch computation deadline, while the light histogram represents the number of task requests that arrived before the batch distribution deadline. The horizontal line is the minimum number of task requests needed to successfully complete the batch. Each result shown is based on trace files from separate days.

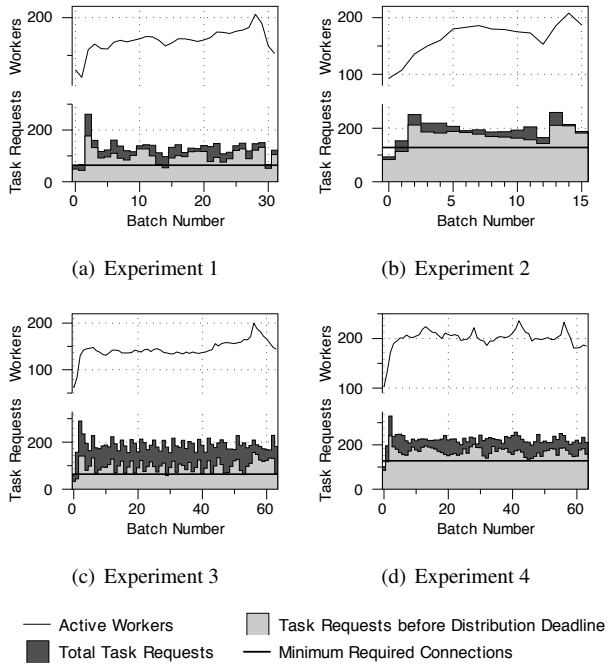


Figure 2. Semi-comm-reliable results

These results show that Algorithm 2 generates sufficient task requests. In all experiments, the algorithm had difficulty during the first few batches due to inaccuracies in counting active workers at the beginning of each experi-

Table 4. Semi-reliable worker simulation

Parameter	Parameter Values
Number of Batches (M)	16, 32
Tasks per Batch (N)	8, 16, 32, 64
Batch Deadline (D)	512, 1024, 2048, 4096
Batch Submission (S_i)	$S_0 = 0; S_{i+1} = S_i + D$
Avg. Task Completion Time ($\overline{R}$)	256 seconds
Task Replication (H)	2, 3, 4 copies

ment. Figure 2(c) shows that batches with relatively short deadlines have trouble in receiving enough task requests. This is likely because task execution with a short deadline is highly sensitive to even short unavailability intervals. Another point worth noting is the sudden lack of task requests during batch 30 in Figure 2(a). This is because of a sudden drop in active workers, and demonstrates a fundamental weakness in low latency batch computing with semi-reliable workers. This weakness is that unexpected drops in active workers can cause missed deadlines. From these results, we feel confident that Algorithm 2 and the model described in Section 2.2 are useful and valid for ensuring sufficient task requests to meet low-latency batch deadlines in VC systems.

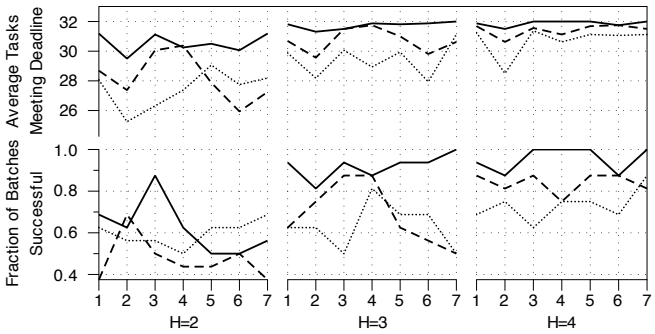
4.2 Semi-Reliable Workers

In these experiments, we used a variety of parameters to test the efficacy of Algorithm 3 in meeting deadlines. For comparison, we used two other algorithms in the simulations, labeled Algorithms B1 and B2. Algorithm B1 is similar to Algorithm 3, except that it computes the connection period as $T \leftarrow (P'L)/(HN)$ instead of using the Poisson model. Algorithm B2 uses a feedback loop to alter the connection period after each batch deadline passes. If any tasks missed the deadline, Algorithm B2 decreases the connection period proportional to the fraction of tasks that missed the deadline. Otherwise, it increases the connection period proportional to the latest task completion time relative to the deadline. Experiments were conducted using 7 separate days of trace data. Two of these days (days 5 and 6) contained high levels of host downtime, which is apparent in the results. Experiment parameters are shown in Table 4.

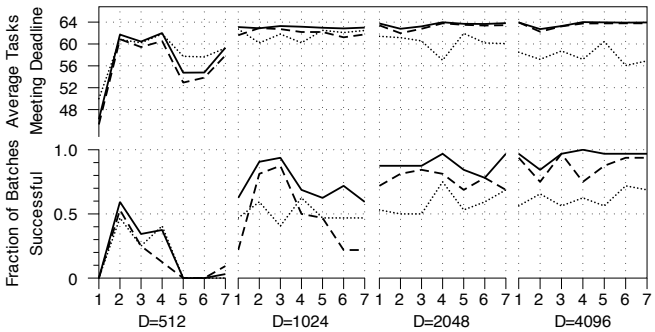
Plots of the experimental results is shown in Figures 3(a) and 3(b). Each figure shows the average number of tasks per batch that met the deadline, and the fraction of batches where all tasks completed before the deadline. The results are plotted for a period of 7 days for each parameter tested.

Figure 3(a) shows the effect of replication in each of the algorithms for replication levels of $H = 2, 3, 4$. It is clear that more task replication improves the effectiveness of all algorithms, but it shows the greatest efficacy for Algorithms 3 and B1. This is likely because the feedback loop algorithm (B2) ignores the fraction of replicas completed on time when deciding how to alter the reconnection period.

Figure 3(b) shows the effect of each algorithm from varying deadline lengths. For a short deadline of $D = 512$, it is natural that all the algorithms fail to perform well. For longer deadlines, all algorithms perform better, but Algorithm B2 seems to hit a plateau. This is likely because a longer deadline means greater fluctuation in the reconnection period as Algorithm B2 attempts to find an optimal period. It is worth noting that in many simulations with short deadlines Algorithm B2 performed best. This is because it quickly drove the reconnection period towards 0, resulting in all hosts constantly connecting to the master server.



(a) Effect of Replication Level on Batch Success (M=16, N=32, D=2048)



(b) Effect of Deadlines on Batch Success (M=32, N=64, H=3)

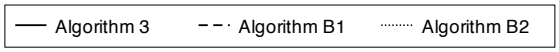


Figure 3. Semi-reliable experiment results

These results demonstrate the effectiveness of using Algorithm 3 for task distribution under soft deadlines. The non-probabilistic Algorithm B1 often performs similarly with high task replication, but fails to take into account host unreliability and therefore consistently underperforms Algorithm 3. The feedback loop based Algorithm B2 underperforms the other two algorithms because it can only react to failed batches, not predict future success. It is interesting to note that high average task success rates (e.g. Figure 3(b), $D = 1024$) don't always translate into high batch success rates because a single late task can delay an entire batch.

5 Related Work

Previous work has studied task distribution in pull-style grid and VC environments [6] [8]. Our algorithms differ from prior research in that they use reconnection requests to semi-reliable workers in order to satisfy soft deadlines.

There has been similar work in regards to completing batches of tasks in desktop grid environments [10]. This study viewed the model of batches as a buffer with tasks periodically entering and expiring from the buffer. The authors analyzed the appropriate buffer size to ensure maximum task completion rates in a desktop grid environment, where tasks can be assigned to arbitrary workers. Future work could investigate means of combining this buffering mechanism with reconnection times.

One technique for handling unreliable workers is the use of checkpointing or "heartbeats" to the master server [7]. In this technique, workers periodically report and/or save their progress to the master server. This allows for duplication of tasks which are unlikely to meet the deadline. However, we believe that this would not be useful in low latency because of the short task short computation time.

6 Conclusion and Future Work

In this paper we proposed models for semi-communication-reliable and semi-computation-reliable workers in a volunteer computing environment. These models were used as a base for several task distribution algorithms aimed at low latency batch VC. The algorithms were validated using execution availability trace data.

This study dealt only with environments containing between 100-200 active workers. In actual VC systems the active worker count can reach hundreds of thousands. In this case, Algorithms 2 and 3 would send reconnection times on the order of days or weeks. This conflicts with worker attrition rates and thus the algorithms would likely fail in such a scenario. Future work could focus on schemes to group workers and perform simultaneous batches.

The current model for task replication is overly pessimistic and could be improved. In future work, we also intend to better integrate the task completion model with Algorithm 3 for more intelligent task distribution. Finally, to make practical use of these results we plan to to study methods for integrating low-latency batch computing in existing volunteer computing systems such as BOINC [2].

Acknowledgments

This work was supported in part by Research Fellowship (19·55401), and Grant-in-Aid for Young Scientists (B)(18700058) and Grant-in-Aid for Scientific Research (B)(18300009) from the Japan Society for the Promotion of Science, and by the Global COE Program "in silico medicine" at Osaka University.

References

[1] Xtremlab web site - http://xw01.lri.fr:4320/, October 2007.

[2] D. P. Anderson. BOINC: A system for public-resource computing and storage. In R. Buyya, editor, *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*, pages 4–10. IEEE Computer Society, 2004.

[3] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[4] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *CCGRID*, pages 73–80. IEEE Computer Society, 2006.

[5] T. W. Anderson and D. A. Darling. Asymptotic theory of certain goodness of fit criteria based on stochastic processes. *Annals of Mathematical Statistics*, 23:193–212, 1952.

[6] K. Budati, J. Sonnek, A. Chandra, and J. Weissman. Ridge: combining reliability and performance in open grid platforms. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 55–64, New York, NY, USA, 2007. ACM Press.

[7] D. Kondo, F. Araujo, P. Domingues, and L. Silva. Validating desktop grid results by comparing intermediate checkpoints. Technical Report TR-0059, Institute on System Architecture, CoreGRID - Network of Excellence, October 2006.

[8] D. Kondo, A. A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *SC'2004 Conference CD*, Pittsburgh, PA, Nov. 2004. IEEE/ACM SIGARCH.

[9] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Resource availability in enterprise desktop grids. *Future Generation Computer Systems*, 23(7):888–903, 2007.

[10] D. Kondo, B. Kindarji, G. Fedak, and F. Cappello. Towards soft real-time applications on enterprise desktop grids. In *CCGRID*, pages 65–72. IEEE Computer Society, 2006.

[11] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Modern Methods in Computational Biology, Horizon Press*, 2003.

[12] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the simgrid simulation framework. In *CCGRID*, pages 138–145. IEEE Computer Society, 2003.

[13] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Euro-Par*, pages 432–441, 2005.

[14] J. M. Schopf and F. Berman. Stochastic scheduling. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 48, New York, NY, USA, 1999. ACM Press.

[15] M. A. Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of American Statistical Association*, 69(347):730–737, 1974.