Resource Co-Allocation : A Complementary Technique that Enhances Performance in Grid Computing Environment

Driss Azougagh* Jung-Lok Yu* Jin-Soo Kim§ Seung-Ryoul Maeng*
Division of Computer Science, Department of EECS,
Korea Advanced Institute of Science and Technology (KAIST), South Korea
*{driss,jlyu,maeng}@calab.kaist.ac.kr §jinsoo@cs.kaist.ac.kr

Abstract

This paper introduces an Availability Check technique (ACT) as a complementary technique to most resource co-allocation protocols in the literature. For a given resource co-allocation protocol, ACT tries to reduce the conflicts that happen between co-allocators when they try to allocate multiple resources simultaneously. In ACT, each job checks for the availability state of required resources and gets informed with updates each time one of the resources availability state changes until all the resources become available. Once all required resources become available a job starts applying the given resource co-allocation protocol. Two co-allocation protocols: All-or-Nothing (AONP) and Order-based Deadlock Prevention (ODP^2) Protocols are chosen to be the case studies to simulate the proposed technique (ACT). To simulate ACT, each job is allowed to allocate from 1 to 5 different types of resources simultaneously using a uniform distribution. The results show that applying ACT to one of the two protocols outperform the original one. The resource utilization is improved by up to 34% and 41% for AONP and ODP^2, respectively. The job response time is improved by up to 13% and 8% for AONP and ODP^2, respectively. And the communication overhead is improved by up to 96% and 94% for AONP and ODP^2, respectively. Also, applying ACT to AONP represents the most scalable and fully distributed scheme that outperforms original schemes (AONP and ODP^2) and competes well with all other schemes presented in this paper.

1. Introduction

Computing over the Internet is becoming increasingly popular. Also, with the emerging infra-structures such as Computational Grids and Web Services, it is possible to develop applications that support various Internet-wide collaborations through seamlessly harnessing appropriate Internet resources. Grid computing has emerged as an important new field [10], distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. Jobs running in computational Grid may require multiple types of resources simultaneously [9]. These observations present new challenges in designing efficient resource allocation protocol that allows an application to simultaneously acquire multiple resources from distributed locations [5], by communicating (or negotiating) only with the corresponding resource managers and without requiring communication with other applications. To overcome these challenges, we argue that a more tighten collaboration between resource managers and resource co-allocators need to be developed. Here, we introduce a new collaborative technique that can easily merge with most existing resource co-allocation protocols.

A distributed system consists of a set of sites that communicate with each other by sending messages over a communication network. We assume that every site has some resources that might be requested by any process from the same or different site. Each site has the capability to exchange light messages with any process requesting resources (from the site). To access resources, a process p_i need to receive permissions from a set of sites S_i. If all sites in S_i grant permission to p_i, then it is allowed to access the resources. To ensure mutual exclusion the sets S_i are required to satisfy the intersection property: for any i and j, $S_i \cap S_j \neq \phi$ when $i \neq j$. These and related concepts were formalized and analyzed in different papers [12, 13, 14, 16].

In general, most introduced multiple resource co-allocation protocols in the literature can be inferred from the *dinning philosophers* problem [3, 7, 13, 16]. The problem consists of five philosophers sitting at a table who do nothing but *think*, *pick up* (sticks), *eat*, and *drop down* (sticks). Between each two neighbor philosophers, there is a single stick. In order to eat, a philosopher must pick up both sticks.

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 IEEE

In this paper, two protocols: All-or-Nothing (AONP) and Order-based Deadlock Prevention (ODP^2) protocols are used as two case studies. In AONP, each philosopher tries to pick up both sticks. If the philosopher picks up one stick or none then he must drop any stick he holds back to the table and after some while he can retry again. In ODP^2 [12, 13, 14], the sticks are supposed to have some global order: by fixing a different number from 1 to 5 for each stick. Each philosopher tries to pick up and secure the sticks one by one starting by the stick that has a small number.

In this work, we propose an additional technique to reduce the time consumed by all philosophers while applying any chosen resource co-allocation protocol. In contrast to the standard formulation of *dinning philosophers* problems which try to focus on either maximizing the number of philosophers who eat immediately or minimizing time it will take to feed all philosophers, our technique tries to cover both of the issues at the same time. This can be achieved simply by adding an alarm for each stick. Each alarm keeps informing its corresponding (or related) philosophers about the availability state of the corresponding stick. When the philosopher realizes that both required sticks are available, the philosopher goes for the action and starts applying the given protocol (AONP or ODP^2). Hence, by using this technique, the philosopher will try not to pick up the sticks if one or both of them are still in use by someone else.

In our proposed technique, the *pick up* stage introduced above in the dinning philosopher problem can be either defined or replaced with an inner loop in which every philosopher must apply before proceeding to the *eating* stage: (1) wait until all stick states are set to *available*, (2) try to *pick-up* sticks, (3) if failed, *drop down* holden sticks and restart from 1, and (4) if succeed proceed to the next (*eat*) stage.

In this paper, we present the technique that enhance the existing resource co-allocation protocols and demonstrate its potential performance benefits using simulation. Next section (section 2) introduces a related work related to resource allocation enhancement presented in the literature. Section 3 introduces the resource co-allocation problem accompanied with a description of AONP and ODP^2, Section 4 describes the proposed technique in details, and Section 5 presents preliminary simulation results. Finally, we conclude this paper with a discussion in Section 6 followed by a conclusion in Section 7.

2. Related Work

Resource management architecture for resource co-allocation protocol is proposed in [6, 8], implemented in Globus Toolkit [1], developed in Condor [17] as a Directed Acyclic Graph Manager (DAGMan), and supported in Legion [4] as a mechanism similar to an atomic transaction strategy. Another way of resource allocation is introduced in Nimrod/G [2] by introducing a Grid Architecture for Computational Economy (GRACE) and proposing simulation model in [11]. In all these researches, the developed resource co-allocation protocols luck improvement and/or scalability when many jobs running in computational Grid require multiple types of resources simultaneously.

Based on general models of computation Grids and parallel tasks running in computational Grids, a parallel task scheduling algorithm for resource co-allocation (PT-SARC) is developed in [18, 19]. It extends the conventional list scheduling [15] heuristics and considers resources co-allocation for parallel tasks in computational Grids. However, their proposed resource co-allocation does not support dynamic scheduling of parallel tasks and it assumes knowledge of all tasks in advance before running the co-allocation algorithm. Instead, our proposed technique enhances existing resource co-allocation protocols when multiple parallel jobs are running simultaneously, without requiring knowledge of the running tasks a priori.

Resource co-allocation to tasks is an important problem for grid computing systems. As an effective solution, agent coalition formation had become a research hotspot [20]. A resource allocation strategy via agent coalition formation for real-time, dynamic, time-bounded grid computing systems is presented. However, no enhancement of the introduced resource co-allocation protocol is presented.

A scalable, decentralized resource co-allocation protocol that can facilitate the dependable deployment of Internet applications is presented in [13, 14]. In this research, the proposed order-based deadlock prevention protocol with parallel requests (ODP^3) protocol ensures deadlock and live-lock freedom during the resource co-allocation process. At the same time, the protoccol takes advantage of parallelism in making resource allocation requests in order to achieve increased efficiency. ODP^3 protocol requires the use of parallel requests feature, a set of goal states associated to each job in which achieving one of the goals satisfies the condition to run the corresponding job. In contrast to ODP^3, our approach does not require using parallel requests feature and still it presents quite competitive results.

3. Multiple Resource Co-Allocation Protocols

This research is conducted to increase the resource usability by reducing resource waste time, communication overhead, and job response time (job waiting time), for small job inter-arrival time and high number of resource types. In this paper we define: (l) job response time as the time a job spends in a queue before it starts execution, (2) communication overhead as the total number of

messages exchanged between resource co-allocators and resource managers to allocate resources for jobs during the simulation (including ACT communication overhead when ACT is applied), and (3) resource waste time as the sum of critical section intervals where a resource cannot be used by any (competitive) job that acquires the resource. A critical section interval of a resource is defined by the interval its corresponding lock is assigned to a non-running job (a job waiting for some other resources to be allocated).

This paper considers a set R of distinct type of resources $R = \{r_1, \ldots, r_m\}$ and a set of jobs $J = \{j_1, \ldots, j_n\}$. A size function $|.|$ can be defined such that $|r|$ represents the total number of instances (or total size) of the resource r and $|j, r|$ represents the number of instances (or size) of the resource r required by the job j. Without lose of generality, for any subset of resources $R' \subset R$ and any subset of jobs $J' \subset J$, we define $|R'|$, $|J', r|$ and $|J', R'|$ as follow:

$|R'| = \sum_{r \in R} |r|$,
$|J', r| = \sum_{j \in J'} |j, r|, and$
$|J', R'| = \sum_{r \in R'} |J', r|$.

We assume that each running job j (or process p_j) tries to get its (required) resources through a resource co-allocator RC_j and each resource r has a resource manager RM_r. When a job j is scheduled, its corresponding resource co-allocator RC_j tries to allocate all required resources in $R_j = \{r \in R / |j, r| > 0\}$ by contacting the corresponding set of resource managers in $SRM_j = \{RM_r / r \in R_j\}$. Once all resource managers in SRM_j reply with success to the resource co-allocator RC_j, the corresponding job j resumes its execution and starts using the resources in R_j.

There are three major problems that degrade the resource usability and job response time and increase communication overhead between the resource co-allocators and the resource managers: deadlock, starvation, and livelock. Deadlock or permanent blocking happens when two or more jobs are trying to allocate two or more resources in a crossing way simultaneously. Starvation might happen when some jobs that are not deadlocked cannot (or will never) proceed due to the competition among jobs to get some resources. Livelock happens when some jobs are forced to release their allocated resources (again and again) when some deadlock prevention technique is used. The described three problems above depend mainly on how much conflict C exist between jobs which can be calculated by the following formula:

$$C(J, R) = \frac{\sum_{r \in R} max(|J,r| - |r|, 0)}{|R| \times |J|}$$

And for this reason, many protocols are presented in the literature [12, 13, 14] that either prevent or detect the occurrence of the above problems.

In this paper we will concentrate our work on two types of protocols, AONP and ODP^2 protocols. In the next two paragraphs, we will present some more details about the two protocols selected for our simulation.

3.1. All-or-Nothing Protocol

In this protocol, the idea is very simple, if a resource co-allocator RC_j fails to allocate some resource r then the resource co-allocator RC_j will release all the resources in R_j it holds and try again sometime later. Once all resources in R_j are allocated successfully, the corresponding job j will proceed and start using the resources in R_j. This protocol is in some how totally decentralized and it does not depend on some global (or centralized) approaches. In this protocol, resource co-allocators use only local information to make their decisions to either allocate or release resources.

Even if each individual parallel computer is reasonably reliable and well understood, the probability of resource re-allocation, sub-job failure due to improper configuration, network error, authorization difficulties, etc., increases rapidly as the number of sub-jobs increases. So, the strict "all or nothing" semantics of the distributed job abstraction severely limited scalability in the real world. Hence, in this paper we introduce a technique that attenuates these problems by reducing the amount of unneeded communication when the number of jobs and the degree of conflict between jobs are quite high. Also, we show that applying our technique to AONP outperforms the original ODP^2.

3.2. Order-based Deadlock Prevention Protocol

To prevent deadlock and to reduce the degree of starvation of resource allocation, this protocol was proposed in expense of some global assumption: distinct resources need to be globally ordered. This assumption makes this protocol neither scalable nor fully decentralized to some extent. This protocol simply applies a multiple requests feature to globally linear ordered resources. The order-based deadlock prevention protocol (ODP^2) requires each job (or the corresponding co-allocator) to secure its resources one by one in an increasing order according to the given global order. Once all resources are allocated the job starts running. This protocol is deadlock-free, starvation-free and livelock-free.

The weak points in this protocol are: resource usability, scalability, and central point of failure - a central node is required to keep global order among resources. The resource usability problem can be seen when a resource co-allocator RC_j secures a resource r and waits for the next resource r' which is still in use by some job j'. During this waiting time all jobs in $J" = \{j"_1, , j"_l\}$ that try to allocate the resource r must wait for the job j, even if some jobs in $J"$ has their next resources are available. To attenuate the degree of this problem, in ODP^2, the job j releases all secured resources and retries some while later when one of the resources cannot be secured for some period of time. An other way to prevent this problem can be achieved by applying our proposed technique. This is because our technique tries

not to let the RC_j to compete for the resources until all required resources in R_j are available at once. By granting this, the number of jobs in J" will reduce and hence the resource usability will increase.

4. Proposed Technique

The proposed technique is based on increasing the cooperation between *resource co-allocators* and *resource managers* over different sites, which we will call as the *Availability Check* Technique (ACT). When a job j tries to allocate resources, the resource co-allocator RC_j sends a request to each resource manager in the set of resource managers SRM_j. Once the resource manager RM_r receives for the first time the request from the resource co-allocator RC_j, it creates a new entry of the job j, computes the availability state of the resource r regarding the job j, and then sends the availability state to the RC_j.

Whenever a change occurs in the resource r, either by locking or releasing r, RM_r sends notifications to the affected jobs in the list of entries created before. If some of the resources in R_j are still unavailable, RC_j keeps receiving updates from all resource managers in SRM_j that experience some changes in their resources regarding the job j. Once all resources in R_j become available at once, RC_j proceed to apply one of the selected resource co-allocation protocols like the ones introduced in the previous section.

Finally, if the job j completes its execution, the job j tries to release the resources in R_j through the corresponding resource co-allocator RC_j by sending a release message to all resource managers in SRM_j. When the resource manager RM_r receives a release request from RC_j, it deletes the corresponding entry of the job j from the list, updates the remaining entries, and notifies those resource co-allocators that have their corresponding states have been changed.

In multiple resource allocation environments, ACT guarantees the improvement of the resource usability by reducing the resources waste time, especially, when the conflict degree $C(R, J)$ is very high. It also guarantees the reduction of communication overhead and job response time.

5. Experimental Results

Here, we present the advantages of the introduced ACT technique by comparing the results of each resource co-allocation protocol when ACT is and is not applied. In the experiment, the jobs compete for 10 different resource types that are distributed. For each simulation run, the maximum capacity of each resource type is set to 1, each job may require up to 5 different resource types (random value from 1 to 5), and the resource usage time is chosen to be a random value between 10 and 1000 seconds to reflect the diversity of the jobs.

For our experiments, we assume that the delay for one-way message delivery follows a uniform distribution with parameters $\frac{RTT}{20}$ and $\frac{RTT}{2}$, where RTT represents the estimated RTT (Round-Trip Time) upper bound for which we choose 2 seconds. We compare the protocols with and without applying the ACT by examining the effect of varying the jobs arrival rates on the resource waste time, communication overhead, waiting time and total completion time. We measured the resource waste time by extracting the total job usage from the total resource usage during the simulation. We define resource waste/usage ratio as the ratio between resource waste time and total job usage ($\frac{TotalResourceUsage - TotalJobUsage}{TotalJobUsage}$). The communication overhead represents the total number of exchanged messages between the resource co-allocators and resource managers required during the whole simulation (including ACT communication overhead when ACT is applied). And the waiting time of a job is measured by the time between its arrival and successful resource co-allocation (starting of execution). We used an exponential distribution to simulate the inter-arrival times of jobs, and varied the mean inter-arrival time from 1000 seconds to 10 second. The simulations were repeated 100 times for each case.

5.1. All-or-Nothing Protocol

In AONP, all results show that applying ACT to AONP achieved better improvement than the original AONP. Figure 1, 2, 3 and 4 show the results of resource waste/usage ratio, communication overhead, average response time (per job), and their improvements accompanied with that of the total completion time, respectively. The workload selected in this experiment was fixed to 64 jobs. The improvement of the resource waste time increases from 3% to 34% as the inter-arrival mean time decreases from 1000 to 10 seconds. In both AONP and ACT-AONP, the total exchanged messages increases as the inter-arrival time becomes smaller. Applying ACT improves the total exchanged messages from 92% to 96% as the inter-arrival time decreases. Once the inter-arrival time goes below 300 seconds the average job response time improvement increases starting from 1% and saturates at 12% (for 64 jobs). The total completion time improvement is in somehow linearly proportional to that of the job average response time and it reaches 8%.

The above results show that applying ACT to AONP may greatly improves the overall system when competitive jobs that compete for multiple resources simultaneously are numerous and the network traffic is severe as it is the case in most distributed systems. This is because ACT tries to prevent jobs from competition when one of its resources is not available. And by reducing the number of jobs competitors, the communication overhead decreases as well.

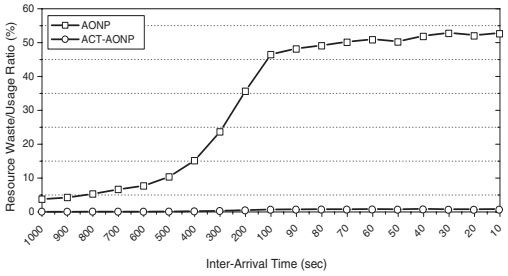

Figure 1. AONP - Resource Waste/Usage

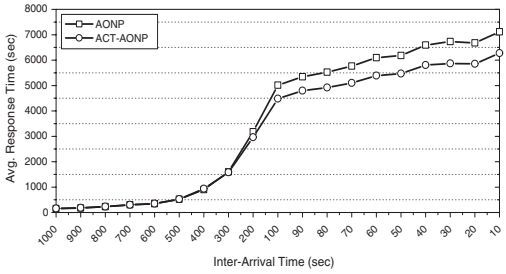

Figure 3. AONP - Average Response Time

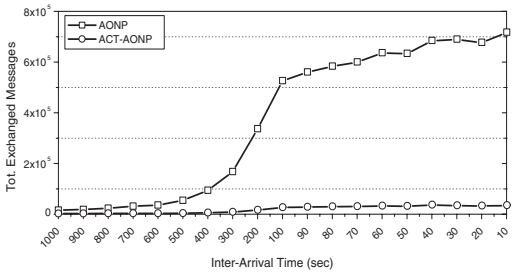

Figure 2. AONP - Communication Overhead

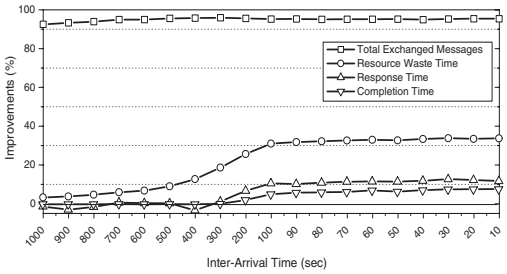

Figure 4. ACT-AONP/AONP - Improvements

5.2. Order-based Deadlock Prevention Protocol

In this section, we introduce the results of applying ACT to two different ODP^2 schemes. The first one is named ODP^{2}I and represents an ODP^2 with infinite timeout. In ODP^{2}I, a job secures a resource once and only once until all it successors are secured and then completes its execution. The second scheme is named ODP^{2}L and has a limited timeout which in these experiments has been fixed to RTT. Once a job fails to lock the current resource at hand before a period of time (RTT) it releases all secured resources and retries after some while.

In these experiments, the workload selected was fixed to 128 jobs for five schemes: ODP^{2}I, ODP^{2}L, ACT-ODP^{2}I, ACT-ODP^{2}L, and ACT-AONP as well. Notice that the results shown in the figures 5, 6 and 7 use the notation "ODPP" instead of the term "ODP^2".

The figures 5, 6 and 7 present results of resource waste/usage ratio, communication overhead, and average job response time, respectively, of the five schemes (ODP^{2}I, ODP^{2}L, ACT-ODP^{2}I, ACT-ODP^{2}L, and ACT-AONP). ODP^{2}I and ACT-ODP^{2}I achieve high degradation in performance concerning resource waste time and average job response time. In contrast, the total exchanged messages when using ODP^{2}I represents the lower bound over all other co-allocating schemes. This can be explained by the fact that, in ODP^{2}I, each job waiting for a resource secures, uses, and releases that resource once only. Overall, in all experiments, applying ACT to resource co-allocation protocols achieves better results.

By looking at figures 4 and 8 we can easily observe the similarity of the improvement results when ACT applied to either AONP or ODP^{2}L. In figure 8, the improvement of the resource waste time increases from 5% to 41% as the inter-arrival mean time decreases from 1000 to 10 seconds. Applying ACT improves the total exchanged messages from 84% to 94% as the inter-arrival time decreases. Once the inter-arrival time goes below 300 seconds the average job response time improvement increases starting from 2% and saturates at 8%. The total completion time improvement is in somehow linearly proportional to that of the average job response time and it reaches up to 3%.

ACT-AONP has been introduced in these experiments to compare its results to those of ODP^{2}L and ACT-ODP^{2}L. The results show that ACT-AONP achieves quite competitive improvements to that of ACT-ODP^{2}L in most results.

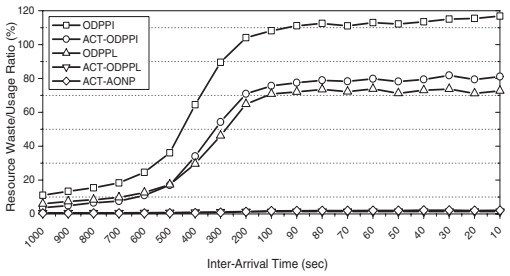

Figure 5. ODP^2 - Resource Waste/Usage

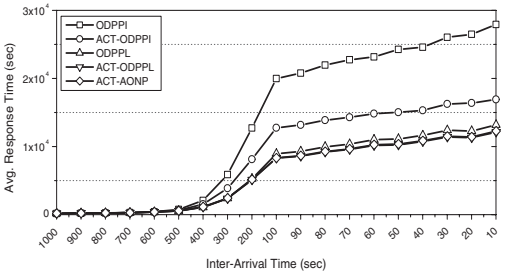

Figure 7. ODP^2 - Average Response Time

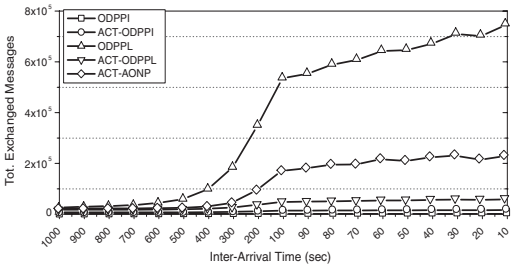

Figure 6. ODP^2 - Communication Overhead

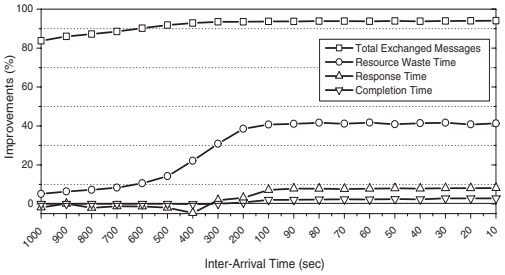

Figure 8. ACT-ODP^{2}L/ODP^{2}L - Improvements

Figure 6 shows that ACT-AONP achieves an improvement of 90% over ODP^{2}L and a degradation of 80% over ACT-ODP^{2}L. This degradation can be seen as a tradeoff over the scalability and resource usability while supporting a fully decentralized protocol when allocating multiple resources. Figure 9 shows the improvements achieved by ACT-AONP over ODP^{2}L. In these results, The response time and the completion time improvements tend to convert to 7% and 1%, respectively, when high inter arrival time is applied.

Fairness analysis shows no difference between the results of ACT-AONP, ODP^{2}L, and ACT-ODP^{2}L. While worst results has been achieved by ODP^{2}I and ACT-ODP^{2}I. Also, the experiments show that the improvements in figure 4 and 8 tend to convert to some limits when the number of jobs increases.

6. Discussion

The figures clearly illustrate the advantage of using the ACT technique regarding the resource usability, communication overhead, and job response time.

ACT reduces the communication occurred between the resource co-allocators and the resource managers mainly in the case of AONP and reduces the number of secured resources that are not in use mainly in the case of ODP^2.

Since the resource usage time for each job is between 10 and 1000 seconds, when the inter-arrival time goes higher (1000 sec) the conflict between jobs decreases and the job response time and total completion time become small. Therefore the effect of ACT, which can be seen in the job response time improvement figures, either is negligible or represents an overhead. When the inter-arrival time goes smaller (10 sec) the conflict between jobs increases, the job response time and completion time become high. Therefore the effect of ACT becomes important. As a conclusion, ACT reduces the number of useless messages to be sent and the number of locked resources which are not in use. These two advantages reduce communication overhead and resource waste time, hence the job response time as well. The only disadvantage of ACT is when the inter-arrival time is very high, the overhead of applying ACT may has some impacts on real distributed systems which can be compromised with the amount of resource waste time and communication overhead reduced (when ACT is applied).

ACT-AONP outperforms the original AONP and ODP^2 protocols. Also, the results of ACT-AONP are quite com-

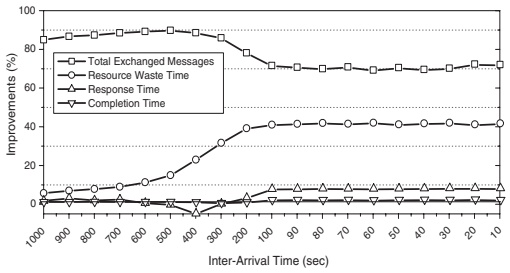

Figure 9. ACT-AONP/ODP^{2}L - Improvements

petitive to those of ACT-ODP^{2}L (as well as ACT-ODP^{2}I). Therefore, ACT-AONP represents a fully decentralized and scalable resource co-allocation protocol that outperforms previous schemes.

7. Conclusion

This discussion clearly shows that the use of ACT improves the resource co-allocation protocols. Up to 12% of overall improvement is achieved in AONP protocol. And in ODP^2, the overall completion time is improved by up to 3% and the job response time is improved by up to 8%. ACT-AONP shows competitive results to those of ACT-ODP^2. In this paper, ACT-AONP is considered to be the most distributed and scalable scheme that has smaller resource waste time and communication overhead among all other schemes.

Since this study shows some interesting results different from the previous research on resource co-allocation protocols, more surveys are needed to generalize the above conclusions for other protocols and to provide more materials to use as a feedback in the future.

References

[1] Grid project. *http://www.globus.org*.

[2] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. *In Proceedings of the 4th International Conference on High Performance Computing*, 1:283–289, May 2000.

[3] K. Chandy and J. MiSra. The drinking philosophers problem. *ACM Transactions on Programming languages and Systems*, 6(4):632–646, October 1984.

[4] S. J. Chapin et al. The legion resource management system. *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 162–178, 1999.

[5] K. Czajkowski et al. A resource management architecture for metacomputing systems. *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[6] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, 3(6):219–228, August 1999.

[7] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138, 1971.

[8] I. Foster et al. A distributed resource management architecture that supports advance reservations and co-allocation. *In Proceedings of the Seventh International Workshop on Quality of Service*, 31(4):27–36, May 1999.

[9] I. Foster and C. Kesselman. The grid: Blueprint for a new computing infrastructure. *Morgan Kaufmann Publishers Inc*, 1999.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal Supercomputer Applications*, 15(3):200–222, June 2001.

[11] J. Gomoluch and M. Schroeder. Market-based resource allocation for grid computing: A model and simulation. pages 211–218, June 2003.

[12] N. A. Lynch. Upper bounds for static resource allocation in a distributed system. *Journal of Computer and System Sciences*, 23(2):254–278, October 1981.

[13] J. Park. A scalable protocol for deadlock and livelock free co-allocation of resources in internet computing. *Proceedings of the Symposium on Applications and the Internet*, 27(31):66–73, January 2003.

[14] J. Park. A deadlock and livelock free protocol for decentralized internet resource coallocation. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(1):123–131, January 2004.

[15] G. I. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection constrained-heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[16] M. Singhal. Deadlock detection in distributed systems. *IEEE Computer*, 22(11):37–48, November 1989.

[17] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. John Wiley & Sons Inc., December 2002.

[18] L. Wang et al. Resource co-allocation for meta-task in computational grids. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 1, pages 36–42, June 2003.

[19] L. Wang et al. Resource co-allocation for parallel tasks in computational grids. pages 88–95, June 2003.

[20] H.-J. Zhang, Q.-H. Li, and Y.-L. Ruan. Resource co-allocation via agent-based coalition formation in computational grids. *International Conference on Machine learning and Cybernetics*, 3:1936–1940, November 2003.