NSF SCI #0438193: Annual Report for Year 2005

Mobile-Agent-Based Middleware for Distributed Job Coordination

Munehiro Fukuda

Computing and Software Systems, University of Washington, Bothell
email: mfukuda@u.washington.edu

January 21, 2006

Abstract

This annual report presents the PI's research activities conducted for year 2005 on NSF SCI #0438193: Mobile-Agent-Based Middleware for Distributed Job Coordination. This is an RUI project to implement the AgentTeamwork system that dispatches a collection of mobile agents to remote computing sites for coordinating the execution of a user job. The PI, his undergraduate research assistants, and research collaborators from Ehime University have implemented and enhanced AgentTeamwork's mobile agent execution platform, system agents, and language support utilities including the mpiJava API. We have evaluated AgentTeamwork's job coordination performance using a Giga-Ethernet cluster of 24 DELL computing nodes and have shown results in our academic papers including the one to be published from International Journal of Applied Intelligence.

Contents

1 Overview

NSF SCI #0438193: Mobile-Agent-Based Middleware for Distributed Job Coordination is an RUI project to implement "AgentTeamwork" that allows a user to dispatch a job with a collection of mobile agents, each deployed to a remote computing site where it launches a job, monitors the job execution, recovers it at another site upon a crash, and even migrates it to an idle computing node for better performance.

Year 2005 was the first year of our three-year NSF-granted research activities. Starting with the purchase and installation of research equipments, we achieved the following research work:

1. an enhancement of our mobile-agent execution platform that serves as AgentTeamwork's infrastructure,
2. an extension of AgentTeamwork's system agents that can dispatch a job, monitor remote computing resources, and detect agent crashes in their hierarchy,
3. a design of AgentTeamwork's preprocessor as well as a study of code cryptography that translates a java program to the code accepted by AgentTeamwork and encrypted for security purposes,
4. an implementation of the mpiJava API with Java sockets and our fault-tolerant socket library named GridTcp.

In addition to the research activities, this report will give details of the PI's student supervision, publications and budget activities in 2005 as well as his research plan for year 2006.

2 Research Activities

This section introduces the research equipments in use for our AgentTeamwork development, describes AgentTeamwork's system overview, and explains our implementation progress for year 2005.

2.1 Research Equipments

We purchased a Giga Ethernet cluster of 24 DELL computing nodes with the NSF grant in February 2005. As shown in Figure 1, it is now connected through the UW Bothell campus backbone to our Myrinet-2000 cluster of eight DELL desktop workstations that has been already installed in summer 2002. Table 1 summarizes the specification of these two cluster systems. Our intension for this cluster configuration is to simulate low parallelism/tightly-coupled CPU connection versus high parallelism/loosely-coupled CPU connection by combining these two clusters and the department's instructional cluster: (1) using the 2Gbps Myrinet cluster for 8-way parallel computation, (2) using the Giga Ethernet cluster for 16 and 24-way computation, (3) using both for 32-way computation, and (4) adding the 100Mbps/1Gbps 32-node instructional cluster to our systems so as to realize 64-way computation.

Specification	Giga Ethernet Cluster	Myrinet 2000 Cluster
# nodes	24	8
CPU	3.2GHz/1MB cache Xeon	2.8GHz Xeon
Memory	512MB	512MB
HDD	36GB SCSI	60GB SCSI
Bandwidth	1Gbps	2Gbps and 1Gbps
Installation	Rank mounted	Desktop
Purchase	With NSF #0438193/departmental budgets	With PI's start-up money in year 2001

Table 1: Cluster specifications

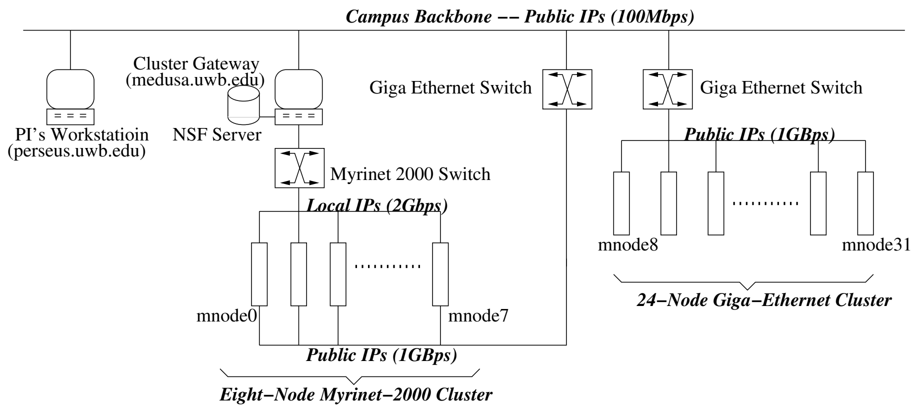


Figure 1: A Configuration of Research Equipments

Java user applications	
mpiJava API	
mpiJava-S	**mpiJava-A**
Java socket	**GridTcp**
User program wrapper	
Commander, resource sentinel, and bookkeeper agents	
UWAgents mobile agent execution platform	
Operating systems	

Figure 2: AgentTeamwork execution layer

2.2 System Configuration

Figure 2 shows the AgentTeamwork execution layers from the top application level to the underlying operating systems. The system facilitates the mpiJava API [6] for high-performance Java applications, while they may call native functions and use socket-based communication as a matter of course. Complying with the original API, AgentTeamwork distinguishes two versions of mpiJava implementation: one is *mpiJava-S* that establishes inter-process communication using the conventional Java socket, and the other is *mpiJava-A* that realizes message-recording and error-recoverable TCP connections using our *GridTcp* socket library. The implementation is user-determined with arguments passed to the *MPJ.Init( )* function. Below mpiJava-S and mpiJava-A is the user program wrapper that periodically serializes a user process into a byte-streamed snapshot. The process execution is coordinated and monitored by AgentTeamwork's system-provided agents such as commander, resource, sentinel, and bookkeeper agents. Each process snapshot is captured by its local sentinel agent and sent to the corresponding bookkeeper agent for recovery purposes. All these agents are executed on top of the UWAgents mobile agent execution platform which we have developed with Java as an infrastructure for agent-based grid-computing middleware.

An application program is coded in the AgentTeamwork-specific framework as shown in Figure 3. If the program uses mpiJava-A, it must include a GridTcp object in a declaration part of system-provided objects (line 4). The code consists of a collection of methods, each of which is named *func_* appended by a 0-based index and which returns the index of the next method to call. The application starts from *func_0*, repeats calling a new method indexed by the return value of its previous method, and ends in the method whose return value is -2, (i.e., *func_2* in this example code). The *MPJ.Init* function invokes mpiJava-A when receiving an ipEntry object that is automatically initialized by the user program wrapper to map an IP name to the corresponding MPI rank (line 8). Following *MPJ.Init*, a user may use any mpiJava functions for inter-process communication (lines 13, 14, 16, and 22). The user program wrapper takes a process snapshot at the end of each function call. Since the GridTcp socket library maintains old MPI messages internally, a process snapshot also contains these messages in it. When the application is moved to or resumed at a new computing node, GridTcp retrieves old messages from the latest snapshot and resends them if they have been lost on their way. Since framework-based programming restricts the programmability of user applications, we are implementing an ANTLR-based language preprocessor that automatically partitions a given Java application into a collection of *func_* methods.

```
 1  public class MyApplication {
 2    public GridIpEntry ipEntry[];        // used by the GridTcp socket library
 3    public int funcId;                   // used by the user program wrapper
 4    public GridTcp tcp;                  // the GridTcp error-recoverable socket
 5    public int nprocess;                 // # processors
 6    public int myRank;                   // processor id
 7    public int func_0(String args[]){    // constructor
 8      MPJ.Init( args, ipEntry );         // invoke mpiJava-A
 9      .....;                             // more statements to be inserted
10      return 1;                          // calls func_1( )
11    }
12    public int func_1( ) {               // called from func_0
13      if ( MPJ.COMM_WORLD.Rank() == 0 )// if I am rank 0, send data
14        MPJ.COMM_WORLD.Send( ... );
15      else                               // otherwise, receive data
16        MPJ.COMM_WORLD.Recv( ... );
17      .....;                             // more statements to be inserted
18      return 2;                          // calls func_2( )
19    }
20    public int func_2( ) {               // called from func_2, the last function
21      .....;                             // more statements to be inserted
22      MPJ.finalize( );                   // stops mpiJava-A
23      return -2;                         // application terminated
24  } }
```

Figure 3: The AgentTeamwork-specific code framework

As shown in Figure 4, a job coordination starts with a commander agent, one of mobile agents provided by AgentTeamwork. This agent creates a resource agent that searches its local XML files for the computing nodes best fitted to the user job's resource requirements. Receiving such candidate nodes, the commander agent spawns a sentinel and a bookkeeper agent. These agents hierarchically spawn as many descendants as the number of nodes required by the job execution. Each sentinel launches a user process and repeats sending its execution snapshot to the corresponding bookkeeper agent. Upon an agent crash, its parent or child agent resumes the crashed agent with the latest snapshot retrieved from the bookkeeper. All results are forwarded through the agent hierarchy from the bottom to the commander agent that thereafter reports them to the client user through the monitor or by files.

2.3 Implementation

Since we have given AgentTeamwork's technical details in our papers [3, 2, 4], this section reports the implementation progress and status of each system components.

2.3.1 UWAgent

UWAgent is a Java-based mobile-agent execution platform that runs at each computing node to exchange AgentTeamwork's agents with other nodes. We completed the initial version in 2004 using Java RMI and verified its functionality not only in our laboratory but also through the PI's senior course project. Submitted from a Unix shell prompt, a new agent forms an agent domain where it recursively forks

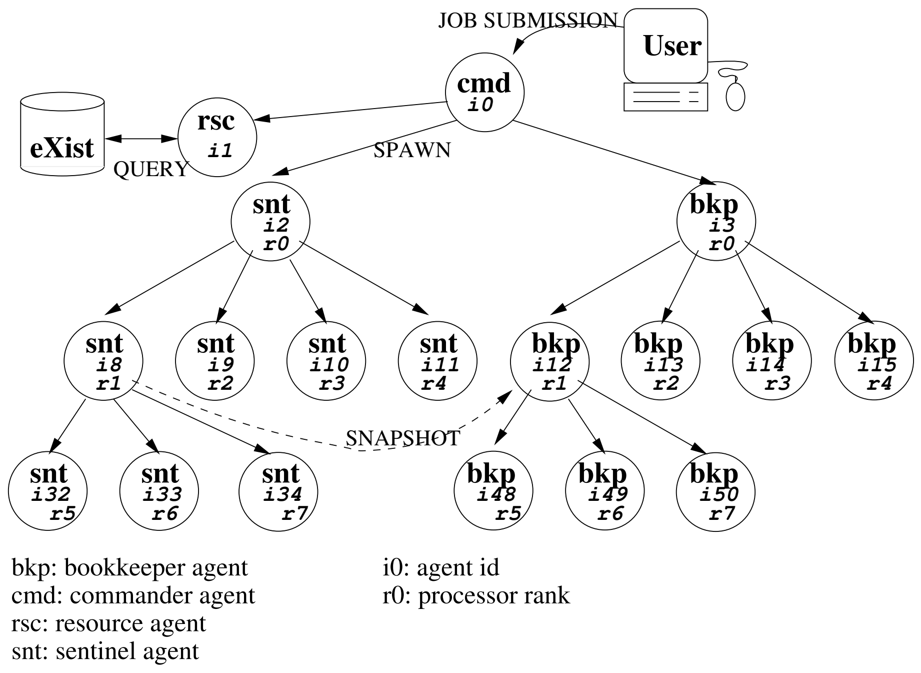


Figure 4: Job coordination in AgentTeamwork

offspring. Therefore, every time a new job is submitted with a commander agent from a Unix shell, it will be executed inside this commander agent's domain, thus without being interfered by other jobs.

However, the initial version has statically limited to 1000 the number of children each agent can spawn. Another restriction is that this version does not allow agents to migrate from a public IP address to a private address domain and vise versa, which in turn means that agents cannot migrate in and out of a cluster system whose computing nodes are accessible with a private IP address from their cluster gateway. We have also noticed that UWAgent's performance depends on Java RMI and its agent migration or communication occasionally takes more than three seconds even between two cluster nodes connected to 1Gbps network.

To address these problems, we reimplemented UWAgent using Java sockets instead of RMI in 2005. The second version now allows a user to specify the number of children each agent can spawn when s/he injects a new agent from the shell. Furthermore, by launching a UWAgent execution platform at a cluster gateway, agents can migrate over the gateway, thus in and out of the cluster. They can also communicate with each other inside and outside of the cluster. We are currently porting all AgentTeamwork's agents to the new version of UWAgent and will soon re-evaluate the performance of agent migration and communication.

2.3.2 AgentTeamwork's System Agents

The following summarizes our incremental development in 2005 for commander, sentinel, bookkeeper, resource, and sensor agents.

Commander Agent

When we submitted our proposal to the NSF middleware program in May 2004 [1], our design of AgentTeamwork used to charge a commander agent with all tasks of spawning and monitoring its descendants such as resource, sentinel, and bookkeeper agents. It was a considerable amount of overhead that prevented AgentTeamwork from scaling up any Java Grande MPJ benchmark programs beyond eight-way parallel computation. A revision has been made to AgentTeamwork by having a commander take care of only each root of resource, sentinel, and book agents that recursively spawns its descendants in a tree structure. This revision has parallelized agent deployment and monitoring work though a tree, whose performance improvement will be demonstrated in Section 3.

Resource Agent

The initial design used to download XML-based computing resource descriptions from a shared ftp server to a Xindice database local to each resource agent. We, however, encountered two problems regarding Xindice: (1) the necessity of a root account for the system maintenance and (2) the complexity of the system porting work, (as pointed out by the NSF review panel.) To address these problems, we changed our local resource database from Xindice to eXist although it is yet an experimental open-source system and thus its documentation is not perfect.

We have enhanced the resource agent's feature so that it can monitor and reflect the status of remote computing resources to its local eXist database. More specifically, a resource agent spawns its descendants named sensor agents in a tree, each repeatedly migrating to a different node in order to sample the availability of its CPU, memory, disk, and network bandwidth. To monitor network bandwidth between any two computing nodes, each sensor in the left half tree performs conventional TTCP communication with its correspondence in the right half tree. Sampled status is periodically reported to the resource agent through the tree. We are currently comparing the accuracy and performance in resource monitoring between our sensor agents and NWS (Network Weather Service) [7], and will enable NWS to start from a sensor as one of AgentTeamwork's user options.

Sentinel Agent

To achieve higher availability and more reliability, we strengthened the sentinel agent's crash detection and recovery algorithm in 2005. Each agent is responsible for checking any crash of its parent and children by periodically exchanging a ping message with them. It resumes a crashed agent by retrieving the latest execution snapshot from the corresponding bookkeeper agent. The algorithm is detailed in our paper [4].

Bookkeeper Agent

We increased the redundancy in maintaining a user process snapshot with multiple bookkeeper agents. Every time a bookkeeper agent i receives a new snapshot from its corresponding sentinel agent, it forwards the snapshot to its neighboring bookkeepers such as agents $i-1$ and $i+1$, so that a crashed user process can be resumed from one of these three bookkeepers even if two of them have been terminated, too.

2.3.3 User Program Wrapper and Preprocessor

As shown in Figure 3, an Java application must be written in or preprocessed into a collection of *func_* methods, each executed and check-pointed by AgentTeamwork's user program wrapper. We are using the preprocessing techniques described in [8]. Focusing on translation from mpiJava to *func_*-based Java programs, we are building two stream template grammars passed to ANTLR: an input grammar for mpiJava language definition and an output grammar for AgentTeamwork's *func_*-based function definition, with which ANTLR automates our language preprocessor design.

We are conducting a feasibility study of the user program wrapper's cryptography that first encrypts a user program on its function basis and dynamically decrypts each function for execution at a remote cite. Our study is currently considering a use of the *CtClass* feature provided by the Javassit project [5] so as to extract a given method from a compiled Java application for cryptography.

2.3.4 GridTcp and mpiJava

GridTcp provides message-recording and error-recoverable TCP connections. (See [1, 2] for its functional details.) As described in Section 2.2, AgentTeamwork provides users with the mpiJava API package that has been implemented in two versions such as Java-socket-based and GridTcp-based implementations, (i.e., mpiJava-S and mpiJava-A respectively). The package includes:

1. *mpjrun.java*: allows an mpiJava application to start a standalone execution (i.e., without using AgentTeamwork) by launching an ssh process at each of remote nodes listed in the "hosts" file.
2. *MPJ.java*: establishes a complete network of the underlying socket connections among all processes engaged in the same job.
3. *JavaComm.java and GridComm.java*: create and maintain a table of Java or GridTcp sockets, each corresponding to a different processor rank.
4. *Communicator.java*: implements all message-passing functions such as *Send( )*, *Recv( )*, *Bcast( )* etc.. It takes care of serializing objects into byte-streamed outgoing messages, exchanging them through its underlying Java or GridTcp sockets, and de-serializing incoming messages to appropriate objects.
5. *MPJMessage.java and Status.java*: return the status of the latest message exchanged.

At the end of 2005, the major mpiJava functions have been implemented and are currently being verified with Java Grande MPJ benchmark programs.

3 Major Findings

In parallel of our AgentTeamwork implementation and enhancement, with the 24-node Giga-Ethernet cluster we have measured AgentTeamwork's computational granularity and scalability as part of our continuous performance evaluation. Since complete results will be published from International Journal of Applied Intelligence [4], this section focuses on only two extreme results.

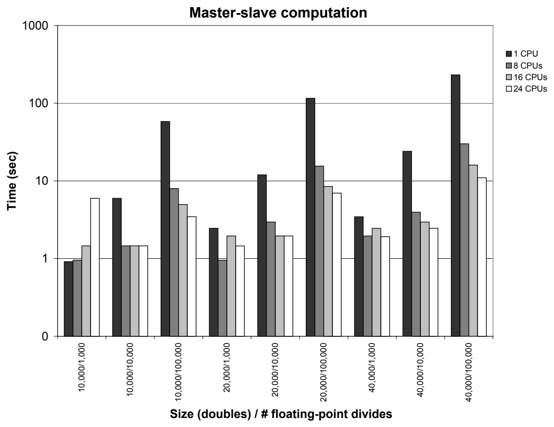
3.1 Computational Granularity

Figure 5 shows mpiJava-A's computational granularity when it has executed our *MasterSlave* and *Broadcast* test programs. Both repeat a set of floating-point computations followed by inter-processor communication and an execution snapshot. The computation is a cyclic division onto each element of a given double-type array. For instance, if they repeat 1,000 divisions onto 10,000 doubles using *P* computing nodes, their computational granularity is $10,000,000/P$ divisions per set. The communication pattern of MasterSlave is data exchange between the master and each slave node, whereas that of Broadcast is to let each node broadcast its entire data set to all the other nodes. In other words, MasterSlave involves the lightest communication, while Broadcast incurs the heaviest communication overhead.

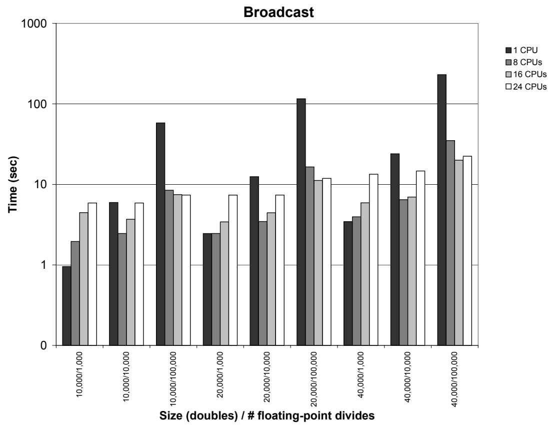
MasterSlave has demonstrated its better parallelism beyond 100,000 floating-point divisions or 40,000 doubles per each communication and snapshot. On the other hand, Broadcast is too communication intensive to scale up to 24 nodes. With 100,000 floating-point divisions, Broadcast's upper-bound is 16 CPUs. We have also coded and run a Heartbeat program where each node exchanges their local data with its left and right neighbors. It has demonstrated computational granularity similar to MasterSlave.

3.2 Computational Scalability

Figure 6 shows AgentTeamwork's computational scalability and overhead factors when it has executed two Java Grande MPJ Benchmark programs: (1) *Series* that computes $40,032/\#nodes$ Fourier coefficients at a different processor and collects all results at the master node, and (2) *MolDyn* that simulates molecular dynamics of 8,788 particles ($8,788 \times 9 = 79,092$ doubles) and exchanges the entire spatial



(A) Master slave



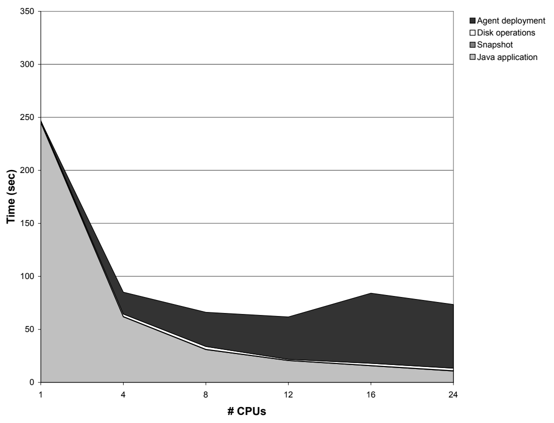
(B) Heartbeat

Figure 5: Computational granularity

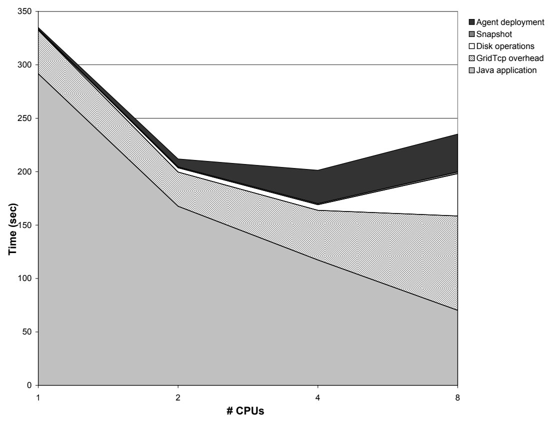
information among processors every simulation cycle. Needless to say, Series and MolDyn represent our MasterSlave and Broadcast granularity test programs respectively.

As shown in Figure 6-A, Series itself is scalable for the number of computing nodes. The largest overhead was agent deployment whose elapsed time was however upper-bounded in logarithmic due to our hierarchical deployment algorithm.

On the other hand, as shown in Figure 6-B, MolDyn has exhibited more overhead in its communication and snapshot-saving operations than agent deployment, which can be characterized in its broadcast communication. This benchmark program unnecessarily forces each node to broadcast the entire collection of spatial data to all the other nodes. We expect that MolDyn will demonstrate its scalable performance on AgentTeamwork, once it is rewritten to direct each computing node to send only its local data to the others or just its left/right neighbors.



(A) Series



(B) MolDyn

Figure 6: Computational Scalability

4 Student Supervision

Through this AgentTeamwork project, the PI supervised eight students in 2005. Their research and programming work is summarized in the table below and more detailed in the following list. The student types shown in the table indicate: (R) an undergraduate research assistant hired with the NSF grant; (E) an Ehime Univ. or UW Bothell exchange student; (U) a CSS undergraduate student working for course credits; and (G) an Ehime Univ. graduate student.

Period (mo/yr-mo/yr)	Students	Type	Research work
01/05 - 06/05	Zhiji Huang	R	implemented mpiJava with Java/GridTcp sockets.
01/05 - 06/05	Enoch Mak	R	implemented AgentTeamwork's resource agent.
06/05 - 02/06	Duncan Smith	R	reimplemented UWAgent with Java sockets.
06/05 - 02/06	Jun Morisaki	E	is implementing AgentTeamwork's sensor agent..
06/05 - 02/06	Jumpei Miyauchi	E	is prototyping AgentTeamwork's GUI.
10/05 - 12/05	Etsuko Sano	U	verified and documented Zhiji Huang's mpiJava.
10/05 - 03/06	Eric Nelson	G	is implementing a language preprocessor.
10/05 - 03/06	Jeremy Hall	E	is working on a function-based code cryptography.

1. **mpiJava Implementation**
   Zhiji Huang, an undergraduate research assistant has implemented the mpiJava API in two versions: mpiJava-S and mpiJava-A. We have reported our implementation techniques and initial performance results in an IEEE PacRim'05 conference paper [2]. Thereafter, Etsuko Sano, a CSS undergraduate student has verified and documented our mpiJava implementation in order to make it available to other students in my senior courses.

2. **Resource and Sensor Agents**
   Enoch Mak, an undergraduate research assistant has ported an Xindice database interface program to eXist, developed an agent-to-node mapping algorithm, and coded the resource agent so that it can pass an arbitrary number of candidate computing nodes to the commander agent. For details of our agent-to-node mapping algorithm, refer to Enoch Mak's final report and the PI's power-point file used for his colloquium talk at Keio University, both available from:
   http://depts.washington.edu/dslab/AgentTeamwork/index.html.

   Jun Morisaki, an Ehime Univ. exchange student has taken over Enoch's work since last June, separated the resource-monitoring feature from the resource agent, and enhanced it in the sensor agent.

3. **UWAgent Mobile Agent Execution Platform**
   Duncan Smith, an undergraduate research assistant has totally reimplemented UWAgent using Java sockets, facilitated agent migration and communication over a cluster gateway, and secured them with SSL.

4. **Language Preprocessor**
   Eric Nelson, a Ehime Univ. graduate student is coding a language preprocessor using the ANTLR tools. Jeremy Hall, a CSS exchange student is working with him at Ehime University to research on the user program wrapper's cryptographic feature that encrypts a user program before its submission and decrypts each function for execution at run time.

5. **GUI and I/O Forwarding**
   Jumpei Miyauchi, an Ehime Univ. exchange student has prototyped AgentTeamwork's GUI with Java applets and is currently developing AgentTeamwork's I/O forwarding mechanism that transfers data files including standard input/output between a user's local directory and his/her remote jobs.

5 Dissemination

We have published two short conference papers and will soon publish one journal paper. The PI has also presented his project status at Keio and Ehime Universities, Japan in 2005. Furthermore, he has contributed to his partner's publication.

5.1 publications

1. Munehiro Fukuda, Koichi Kashiwagi, Shinya Kobayashi, "The Design Concept and Initial Implementation of AgentTeamwork Grid Computing Middleware", In Proc. of IEEE Pacific Rim Conference on Communication, Computers, and Signal Processing - PACRIM'05, pages 255–258 Victoria, BC, August 24–26, 2005
2. Munehiro Fukuda, Zhiji Huang, "The Check-Pointed and Error-Recoverable MPI Java Library of AgentTeamwork Grid Computing Middleware", In Proc. of IEEE Pacific Rim Conference on Communication, Computers, and Signal Processing - PACRIM'05, pages 259–262 Victoria, BC, August 24–26, 2005
3. Munehiro Fukuda, Koichi Kashiwagi, Shinya Kobayashi, "AgentTeamwork: Coordinating Grid-Computing Jobs with Mobile Agents", In Special Issue on Agent-Based Grid Computing, International Journal of Applied Intelligence, to appear in 2006

5.2 Colloquia

1. "AgentTeamwork: Mobile-Agent-Based Middleware for Distributed Job Coordination", Colloquium for the Novel Computing Project & Multimedia Databased Laboratories, Faculty of Environmental Information, Keio University, December 20, 2005
2. "Grid Computing Using Mobile Agents", Colloquium at IEEE Shikoku-Japan Section , Ehime University, December 22, 2005

5.3 Contribution to Partner's Publication

1. Shinya Kobayashi, Shinji Morigaki, Eric Nelson, Koichi Kashiwagi, Yoshinobu Higami, Munehiro Fukuda, "Code Migration Concealment by Interleaving Dummy Segments", In Proc. of IEEE Pacific Rim Conference on Communication, Computers, and Signal Processing - PACRIM'05, pages 269–272 Victoria, BC, August 24–26, 2005

6 Budget Activities

This section reports the PI's budget activities in terms of equipment purchases, his salary for course release, student salary, and expenses for his trips to Victoria, BC and Japan.

6.1 Equipments

The following table details the purchase of our Giga-Ethernet cluster of 24 DELL computing nodes. Since the equipment budget line was cut off to less than a half of our original estimation, we purchased this cluster system in support from the PI's departmental budget. The deficit will be compensated with part of the indirect cost to be returned to the department.

Description	Price	Quantity	Amount
3.2GHz/1MB Cache, Xeon 800MHz Front Side Bus	$762.89	24	$18,309.36
16Amp, Power Distribution Unit120V, w/IEC to IEC Cords	$89.00	2	$179.00
RJ-45 CAT 5e Patch Cable, Snagless Molded - 7 ft	$1.61	28	$45.08
PowerConnect 2624 Unmanaged Switch, 24 Port GigE with 1 GigE/SFP Combo Port	$315.92	1	$315.92
Tax			$1,653.27
Total			$20,501.63
Alloted amount			$9,866.00
Difference			-$10,635.63

6.2 PI's Salary

As shown in the following table, the PI used his research salary for two times of course release, so that his teaching responsibility was reduced to one course per each quarter through academic year 2005.

Quarters	course releases	Salaries
Spring 05	CSS342: Mathematical Principle of Computing	$12,485.00
Autumn 05	CSS430: Operating Systems	$12.982.00
Total		$25,467.00
Alloted amount for 2005		$24,480.00
Difference		-$987.00

6.3 Student Salary

The PI has hired three undergraduate students for their full research commitment to and one more student for his partial involvement in the AgentTeamwork project.

Name	Research Item	Hourly	Working hours	Salaries
Zhiji Huang	mpiJava	$14.00	200hrs in wi05, 120hrs in sp05	$5,264.00
Enoch Mak	Resource agent	$14.00	176hrs in wi05, 204hrs in sp05	$5,320.00
Duncan Smith	UWAgent	$14.00	131.5hrs in Su05, 228.5hrs in au05	$5,040.00
Jeremy Hall	Code cryptography	$14.00	43hrs in au05	$602.00
Total				$16226.00
Alloted amount for year 2005				$16800.00
Difference				$574.00

6.4 Travels

The PI has traveled abroad to Victoria, BC and through Japan. The following table summarizes the expenses of his trips.

Trip (dates)	Tasks	Amount
Two paper presentations at IEEE PacRim 05 (8/24/05–8/26/05)	Victoria registration fee	$331.42
	Victoria registration fee	$163.75
	Ferries to/from Victoria	$119.50
	Encumbered per diem (two nights)	$527.82
Colloquia at Keio & Ehime Universities (12/14/05-1/3/06)	Flights to/from Tokyo Japan	$771.00
	Flight between Tokyo and Ehime	$274.45
	Encumbered per diem (one night)	$257.00
Total		$2,444.94
Alloted amount for year 2005		$2,500.00
Difference		$55.06

7 Plan for Year 2006

In order to accomplish the following research items, the PI is planning to hire three undergraduate research assistants, to supervise a few more students through their research course, (i.e., CSS499: Undergraduate Research), and to keep collaborative work with the Ehime Univ. researchers and graduate students. This year we are targeting three conferences such as PDPTA, Cluster06, and CCGrid for our paper submission.

1. **Inter-Cluster Job Coordination**
   Since the latest version of UWAgent allows agents to migrate over a cluster gateway, we will first port AgentTeamwork to it. Thereafter, we will enhance AgentTeamwork's node allocation algorithm so that a collection of sentinel agents deploy a parallel applications over two or more cluster systems.

2. **Dynamic job Scheduling**
   UWAgent currently facilitates only an FCFS-based thread queue for executing a user job as a thread. We will elaborate our priority-based scheduling policy and implement it in UWAgent. The sentinel agent must be enhanced for its user job coordination so that it will move a job to another computing node whenever finding the job executed in a low priority or under a poor condition that cannot allocate enough computing resources.

3. **GUI and File Transfer**
   In addition to menu-based job submission, AgentTeamwork's GUI should allow users to keep track of their job coordination, (more specifically the activity of their mobile agents) as well as display computation results graphically.

   The I/O forwarding feature should allow sentinel agents to carry with them files that have been opened but not yet closed for read or write whenever the agents migrate to a new location.

4. **Preprocessor and Security**
   We are planning to prototype AgentTeamwork's preprocessor by late March so as to be in time for its use in the PI's senior course, (i.e., CSS434: Parallel and Distributed Computing) from this April. We are also expecting that we will be able to complete our feasibility study and design of the user program wrapper's function-based cryptography by this summer, and to start implementing this feature as soon as possible.

5. **Applications**
   Through the PI's senior courses as well as undergraduate research supervision, we expect to have more applications for testing and evaluating AgentTeamwork.

6. **Resource Database**
   If we have enough manpowers, we will implement our own resource database manager instead of using eXist, which mitigates AgentTeamwork's installation work at each user's computing site.

8 Final Comments

Our research progress for year 2005 was right on the original schedule described in our NSF proposal [1] except two work items: GUI and dynamic job scheduling. Although we initially plan on a GUI implementation in 2007, we prototyped it in 2005 and will complete its feature this year. On the other hand, our algorithm development of dynamic job scheduling was planned in 2005 but had to wait till this year. This is because we thought that the development work would become easier after the enhancement of each AgentTeamwork's component and should be achieved in parallel of our implementation of inter-cluster job coordination.

By completing this year's work items listed in Section 7, we expect that we will be able to focus on application development, system validation and refinements, and dissemination in year 2007, (i.e., our NSF grant's final year) as we originally planned.

References

[1] Munehiro Fukuda. SCI: Mobile-agent-based middleware for distributed job coordination. Proposal to NFS Network Centric Midddlewware Service, #0438193, University of Washington, Bothell, WA 98011, May 2004.

[2] Munehiro Fukuda and Zhiji Huang. The check-pointed and error-recoverable MPI Java library of AgentTeamwork gird computing middleware. In *Proc. IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing - PacRim'05*, pages 259–262, Victoria, BC, August 2005. IEEE.

[3] Munehiro Fukuda, Koichi Kashiwagi, and Shinya Kobayashi. The design concept and initial implementation of AgentTeamwork grid computing middleware. In *Proc. IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing - PacRim'05*, pages 255–258, Victoria, BC, August 2005. IEEE.

[4] Munehiro Fukuda, Koichi Kashiwagi, and Shinya Kobayashi. AgemtTeamwork: Coordinating grid-computing jobs with mobile agents. *International Journal of Applied Intelligence*, to appear in Speciall Issue on Agent-Based Grid Computing, 2006.

[5] Javassist Home Page. http://www.csg.is.titech.ac.jp/ chiba/javassist/index.html.

[6] mpiJava Home Page. http://www.hpjava.org/mpijava.html.

[7] Network Weather Service Home Page. http://nws.cs.ucsd.edu/.

[8] C. Wicke, L. Bic, M. Dillencourt, and M. Fukuda. Automatic state capture of self-migrating computations in MESSENGERS. In *Proc. MA'98*, pages 68–79. Springer, September 1998.