AgentTeamwork

Midterm Report

Enhancing Communication and File I/O

Joshua Phillips
0222418

10/09/06 – 12/15/06

Table of Contents

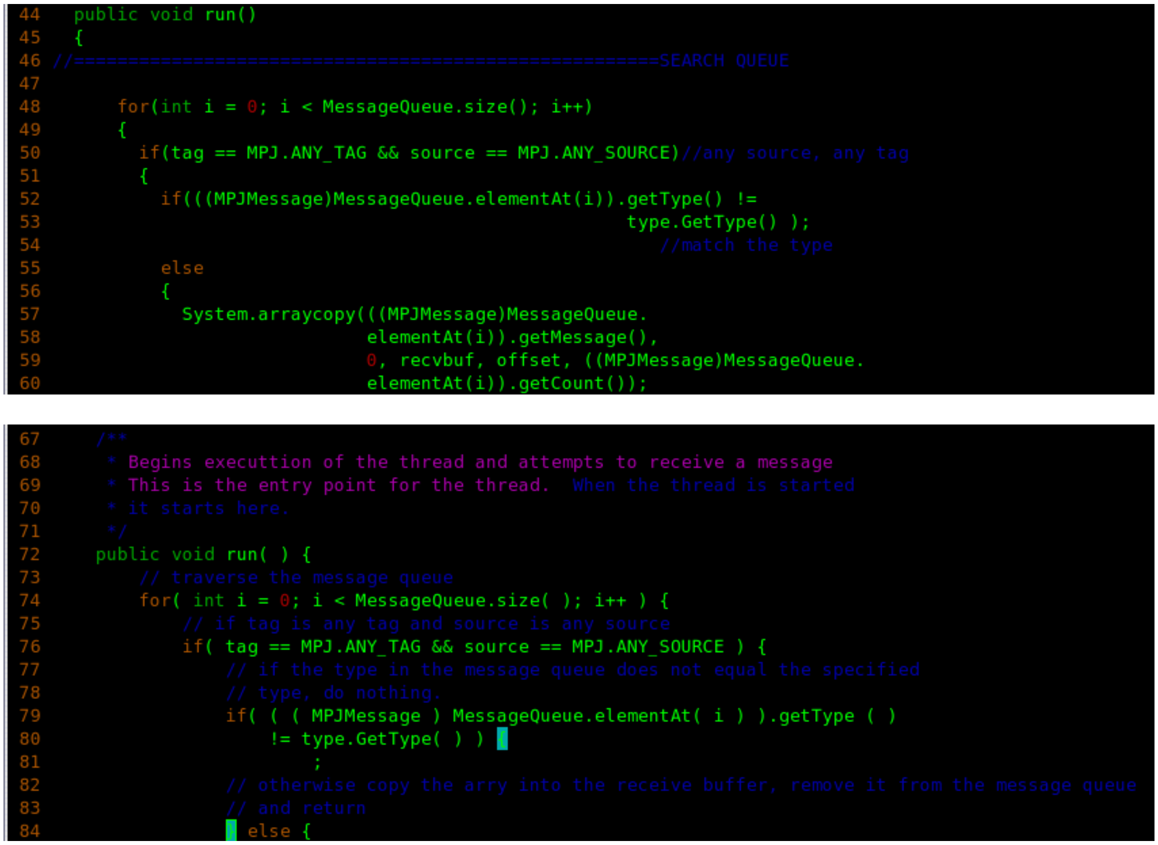
Major Accomplishments

My contributions to AgentTeamwork are divided into seven discrete phases, four of which have been completed. Each of those four phases and the major accomplishments they realized are listed in detail:

Phase 1 - Message Passing Java's Debugging/Code Reformatting/Javadoc

During this phase my primary goal was to debug MPJ, reformat some of its code, and generate javadoc. To increase the readability and maintainability of MPJ I started by adding comments and javadoc tags to the following classes, as well as consistently indenting and aligning brackets: *IRecvThread*, *ISendThread*, *Mpjrun*, and *Request*. In addition to increasing readability and maintainability MPJ this phase provided me a deep understanding of the MPJ package and its inner intricacies. This understanding proved useful in testing MPJ and may also be useful during later phases of the project.

To create the most effective javadoc, I have used Sun's guide to javadoc tags consistently throughout this and all other phases of my contract [1]. In short, that document specifies standards for where to put the most appropriate javadoc tags.



```
  public void run()
  {
//=======================================================SEARCH QUEUE

      for(int i = 0; i < MessageQueue.size(); i++)
      {
        if(tag == MPJ.ANY_TAG && source == MPJ.ANY_SOURCE)//any source, any tag
        {
          if(((MPJMessage)MessageQueue.elementAt(i)).getType() !=
                                                    type.GetType() );
                                                       //match the type

          else
          {
            System.arraycopy(((MPJMessage)MessageQueue.
                             elementAt(i)).getMessage(),
                             0, recvbuf, offset, ((MPJMessage)MessageQueue.
                             elementAt(i)).getCount());
```

```
    /**
     * Begins executtion of the thread and attempts to receive a message
     * This is the entry point for the thread.  When the thread is started
     * it starts here.
     */
    public void run( ) {
        // traverse the message queue
        for( int i = 0; i < MessageQueue.size( ); i++ ) {
            // if tag is any tag and source is any source
            if( tag == MPJ.ANY_TAG && source == MPJ.ANY_SOURCE ) {
                // if the type in the message queue does not equal the specified
                // type, do nothing.
                if( ( ( MPJMessage ) MessageQueue.elementAt( i ) ).getType ( )
                    != type.GetType( ) ) {
                        ;
                // otherwise copy the arry into the receive buffer, remove it from the message queue
                // and return
                } else {
```

Figure 1. IRecvThread's run() method before commenting (above) and after commenting (below)

In addition to reformatting, I debugged the following methods of the *Communicator* class: *Gatherv*, *Scatter*, *Scatterv*, *Allgather*, *Allgatherv*, *Alltoall*, *Alltoallv*, *Bcast*, *Reduce*, *Allreduce*, *Reduce_scatter*, *Isend*, and *Irecv*. To accomplish this I implemented a very simple but extensive test class (*CommunicatorTest*) that uses a master node to pass messages through these methods to a slave node.

The results of the tests were very promising. Originally, it was thought that there may have been some very severe bugs within MPJ to cause frequent OutOfMemoryError exceptions, but upon further research I found that the main issue was merely the default limitations of the Java Virtual Machine. The JVM can be passed an option to specify the maximum amount of system memory on the heap it may allocate. This default is set to 64MB. With such a large amount of array copies and large buffers similar to the ones MPJ uses, the JVM will easily crash. Upon telling the JVM to allocate all available memory space on its node with the argument *–Xmx512MB* nearly all of the OutOfMemoryError exceptions disappeared.

There were, however, a few more bugs that needed attention. I have fixed all but one of these bugs. The *ISend* and *IRecv* had very simple threading bugs that constantly threw IllegalMonitorStateException exceptions but were easily fixed with careful placement of monitor locks through the use of Java's keyword *synchronized*.

Additionally, while *Communicator*'s *Allgather* method was implemented simply by calling *Gather* and *Bcast*, for some reason, *Allgatherv* had a completely new and separate implementation. Unfortunately, this implementation included an additional temporary and unnecessary buffer that for large amounts of data would contribute to consuming all of a node's system memory. The old solution might have been devised to increase the algorithm's speed but I felt that the significant loss of memory far outweighed the small performance gain from the separate algorithm so I re-implemented *Allgatherv* to simply call the *Gatherv* and *Bcast* methods in succession.

Lastly, I discovered a significant bug in MPJ's *Reduce* method that causes OutOfMemoryError exceptions but that I was not able to fix due to time constraints and the severity of the problem. Again, this bug stems from unnecessary temporary buffers. In case of large user-created buffers, it is absolutely fundamental that no additional buffers of a significant size be allocated. For example: in a two-node system in which each node uses a 200 MB send buffer, the master node will have a 400 MB buffer allocated when using the *Gather* method. Fortunately, the master can use the same buffer for sending and receiving data. However, with *Reduce*'s current implementation, an additional 200 MB buffer is created to copy its send buffer's data before any operations are carried out on it. This is done because *Reduce* allows the user to specify an offset for its receive buffer. If the user has decided to use the same buffer for sending and receiving and has specified a receive buffer offset, *Reduce* will overwrite its send contents (used in *Reduce*'s operation) with results of previous operations. To overcome this problem, *Reduce*'s author has allocated a temporary buffer to copy *all* of the send buffer's contents. Because each operation in *Reduce* is completed element-wise, this is unnecessary. I have designed, but not coded, an algorithm that can overcome this shortfall very simply by starting operations at the offset point, storing each result at the end of buffer (just after the last element of the send buffer), and then returning to the beginning of the buffer and performing operations up until the offset point. This will work because all original send values will be preserved up until they are needed, and then overwritten. Also, if an offset is specified in the receive buffer, its length must be greater than or equal to the amount of data sent plus the offset amount. Therefore, there should be enough space to store the operation results correctly. I have chosen not to implement this algorithm yet because of MPJ's architecture. A correct implementation would require me not only to modify *Communicator*'s *Reduce* method, but also each one of *Datatype*'s (e.g. *MPJBool*, *MPJByte*, etc.) operations (e.g. Sum, Product, etc.). That means I would need to modify roughly 96 functions (8 dataypes * 12 operations).

In conclusion, this phase has been completed with the generation of useful javadoc, increased readability and maintainability in certain components of MPJ and a successful run of *JGFPingPongBench* on 2 processors and successful runs of *JGFGatherBench* 2, 4, and 8 processors (see Appendix).

Phase 2 – Ateam/UserProgWrapper Implementation

In short, the purpose of Phase 2 was to create a much simpler framework for users of AgentTeamwork to develop a program within. The answer to this requirement can be summed up in a single class: *Ateam*. *Ateam* is an object that any user program may contain and which provides an intrinsic and transparent *GridFile*, *GridTcp*, *GridIpEntry* table, rank, and size (number of hosts). A user program's *Ateam* object also supplies the user program with additional functionality through the following methods: *takeSnapshot*, *isResumed*, *getSnapshotId*, *registerLocalVar*, and *retrieveLocalVar*.

The *takeSnapshot* method takes a snapshot and stores accepts an ID number to label the snapshot with, *isResumed* tests whether the current user program has been crashed and resumed, and *getSnapshotId* returns the ID number of the current snapshot. The reason for the *registerLocalVar* and *retrieveLocalVar* methods is slightly complex. *Ateam*'s ultimate goal is to make an AgentTeamwork user program as simple and unrestricted as possible. See Appendix B for an example AgentTeamwork user program. Note that a user program is instantiated within the user program class's own static main method. To make AgentTeamwork fully recoverable it is necessary to be able to serialize all of a user program. However, local variables are not serialized. To overcome this problem *Ateam* provides *registerLocalVar* and *retrieveLocalVar* that add and retrieve local variables to a serializable hash table that is stored in every snapshot.

To allow AgentTeamwork to successfully execute an *Ateam* user program, it was necessary to heavily modify *UserProgWrapper*. While I won't go into specifics, the final result is a user program wrapper that will launch and create snapshots of an *Ateam* user program while retaining the old functionality that supports user programs without the *Ateam* class (i.e. – a partitioned program). Halfway through this phase, a new problem arose that required a clever solution. As stated before, it is paramount that the *Ateam* package allows a user program to behave as much like any other java application as possible. To do this, we must allow the user to create their own static main function. In Java, a non-static variable can not be accessed from a static method. As seen in example user program provided in the appendix, that means that a user program's *Ateam* member must be declared static. For obvious reasons, static variables are not included in an object's serialization. The solution was to create an abstract class called *AteamProg* that every AgentTeamwork Ateam user program must extend. This class overrides the default *readObject* and *writeObject* methods that the *Serializable* interface provides. This way, when the user program is serialized it first copies the static *Ateam* member into a non-static member so that it will be included in a snapshot. When a user program object is deserialized, the reverse operation takes place and all of this is completely transparent to the user. In fact, the added benefit of the *AteamProg* class is that it simplifies the user program even more by only requiring them to extend from it. There is no need for a user program to declare its own *Ateam* member or implement serializable because *AteamProg* takes care of both of those requirements transparently.

Additionally, I have created *Socket* and *ServerSocket* classes in the *Ateam* package that merely wrap the *GridTcp* package's *GridSocket* and *GridServerSocket* classes to provide even more convenient and seamless user program implementation and/or porting.

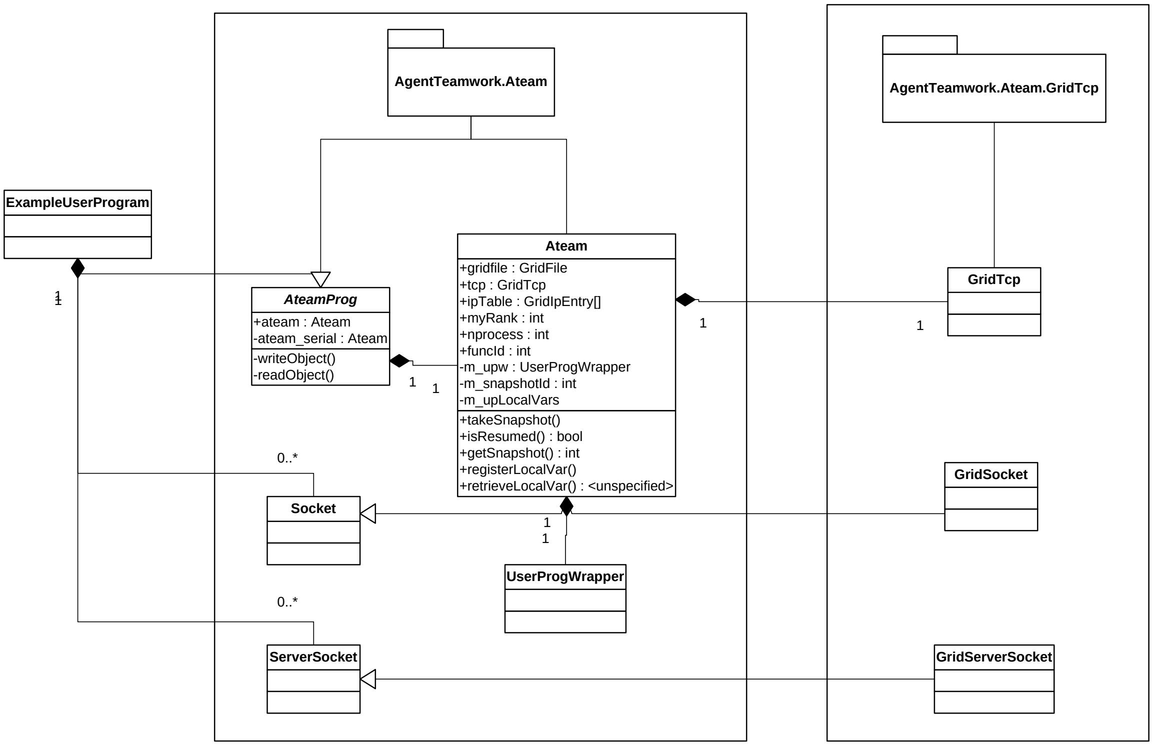


Figure 2. AgentTeamwork.Ateam package UML Diagram

Lastly, to improve my efficiency and that of other AgentTeamwork contributors, during this phase I placed each component of AgentTeamwork into its own package, organized all source files into an easy to understand source-development-tree and created script files to simply the compilation and javadoc generation of AgentTeamwork. The explanation of this source-development-tree can be found in Appendix C. Put simply, AgentTeamwork is now divided into the following packages: UWAgent, MPJ, MPJ.JGF, AgentTeamwork.Agents, AgentTeamwork.Ateam, AgentTeamwork.Ateam.GridTcp, and AgentTeamwork.Ateam.GridFile. This structure completely eliminates the need to jar source files and specify complex class-paths when compiling AgentTeamwork. All compilation can either occur from the root of the source-development-tree or through the simple scripts in the scripts directory.

Phase 3 – GridTcp Revision

Phase 3's purpose was to increase the stability of GridTcp by providing flow control and memory management features to prevent it from causing OutOfMemoryError exceptions. As of the time this document was written this has implementation has been finished, tested, and shown to work properly for the most part. Very late in the JGF tests memory errors still occur when the buffer sizes start to become larger. I am in the process of locating this bug.

Memory Management

Due to its recoverability, GridTcp tends to require large amounts of memory to store all of its backup packets. This can quickly lead to slow node performance and/or out of memory exceptions within GridTcp. To overcome this issue, I have modified the GridTcp package to store older backup packets to disk when memory usage has become too high. Put simply, my solution uses an event-based model and a custom designed DiskVector class to accomplish this.

A GridConnection has four in-memory queues and two on-disk queues to store and manage its packets:

- *Backup queue:*
  If rollback is enabled, each time a packet is sent, it is stored here until a commit message is received
- *Forwarding queue:*
  For forwarding packets through gateways: this queue remains untouched by memory management
- *Incoming queue:*
  Every time a data packet is received it is stored here until it is read by the user
- *Outgoing queue:*
  Regardless of rollback, each time a packet is sent, it is stored here until the correct ACK was received. This queue is also used for flow control. If a client has received a PAUSE message, it will store any outgoing messages here until a RESUME message is received
- *Backup disk queue:*
  Holds older, overflow backup packets when GridTcp has reached its threshold
- *Outgoing disk queue:*
  Holds older, overflow outgoing packets when GridTcp has reached its threshold

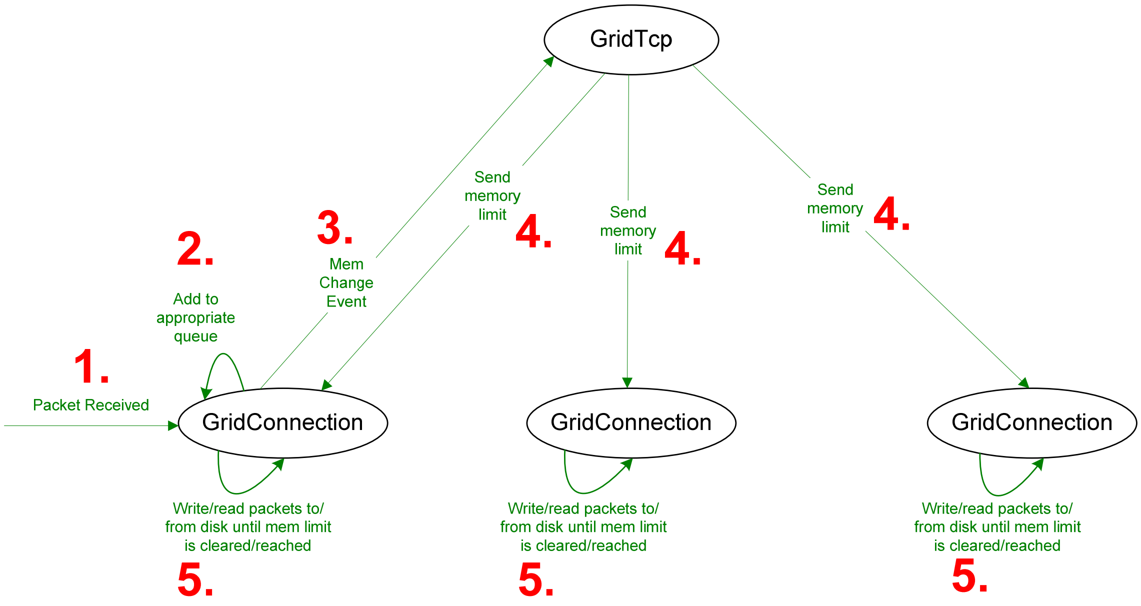

GridTcp
Send memory limit
4.
Send memory limit
4.
Send memory limit
4.
3.
Mem Change Event
2.
Add to appropriate queue
1.
Packet Received
GridConnection
GridConnection
GridConnection
Write/read packets to/ from disk until mem limit is cleared/reached
5.
Write/read packets to/ from disk until mem limit is cleared/reached
5.
Write/read packets to/ from disk until mem limit is cleared/reached
5.

Figure 3. GridTcp's Memory Management Data Flow Diagram

GridTcp manages its memory usage in the following manner:

1. A user set's its threshold (in bytes) via the *setMemThreshold* method
2. When a new GridConnection is created, GridTcp subscribes to its memory change event
3. When a GridConnection adds or removes a packet from any queue, it sends a memory change event to all subscribers, in this case its GridTcp creator
4. Upon receiving this event, GridTcp determines whether or not the collective memory usage of all of it's GridConnection's has surpassed the user-defined threshold
5. If this threshold was previously not reached but is now surpassed it calculates a per connection memory threshold, passes it to each connection, and tells each connection to write old packets to disk until the threshold is cleared

6. If this threshold was previously passed but is now cleared it calculates a per connection memory threshold, passes it to each connection, and tells each connection to read old packets to disk until the threshold is almost reached
7. If at anytime backup or outgoing packets are deleted and/or accessed (i.e. – if a rollback is requested, or a commit is received, or a data ACK is received) both in-memory and on-disk queues are traversed
8. When GridTcp's *disconnect* method is called by a user, each GridConnection is forced to remove any files it has left on disk to minimize disk waste

A few notes about my design:

- Outgoing packets are given in-memory priority over backup packets as it is more likely that they will be retrieved sooner. Also, outgoing packets are more likely to have their contents read than backup packets, which are more likely to just be deleted. It is obviously takes a bigger performance hit to read from disk than it does to read from memory.
- There is currently no feature to remove old backup files from disk if a node has crashed. This is a normal situation however, and because all backup packets are stored in the /tmp folder they can most likely be regularly cleaned by the operating system and or user.
- Using an event-based model adds a small amount of complexity to GridTcp and the addition of a few classes, but the benefits definitely outweigh the drawbacks. By using events, we eliminate the need for GridTcp to constantly poll it's connections for memory usage, and alternately eliminate a circular dependency (or coupling) of GridTcp and GridConnection. If GridConnection needed to call a GridTcp method it would require a reference to GridTcp and these two classes would be useless without each other. Event's eliminate this coupling and provide the ability for future classes to receive notifications of memory changes if need be.
- The previous week's version of GridTcp's memory management used a GridConnBackup file that stored all backed up packets into a single vector on disk. I quickly realized the fault in this solution: to access the vector you must load it entirely into memory first which automatically violates the purpose of the memory management. The new version stores each GridPacket in a separate file on disk, managed by the new DiskVector class.

Flow Control

While GridTcp's new memory management takes care of backup and outgoing packets becoming to numerous, a GridConnection's incoming packets are left unchecked. To alleviate this problem, I have designed a simple flow-control algorithm that limits a server's load as well as the network when its memory becomes to full to accept new packets.

Originally, I started out to design a simple one-to-one packet-to-ACK flow control mechanism that only sent a new package if the previous ACK was received. While this is easy to implement, its performance is far less than perfect. So, I have designed a much more ambitious and complex algorithm that works well but has put me nearly three days behind schedule. This algorithm uses PAUSE and RESUME messages that allows a client to send packets more aggressively but refuses packets when its incoming queue becomes to large.

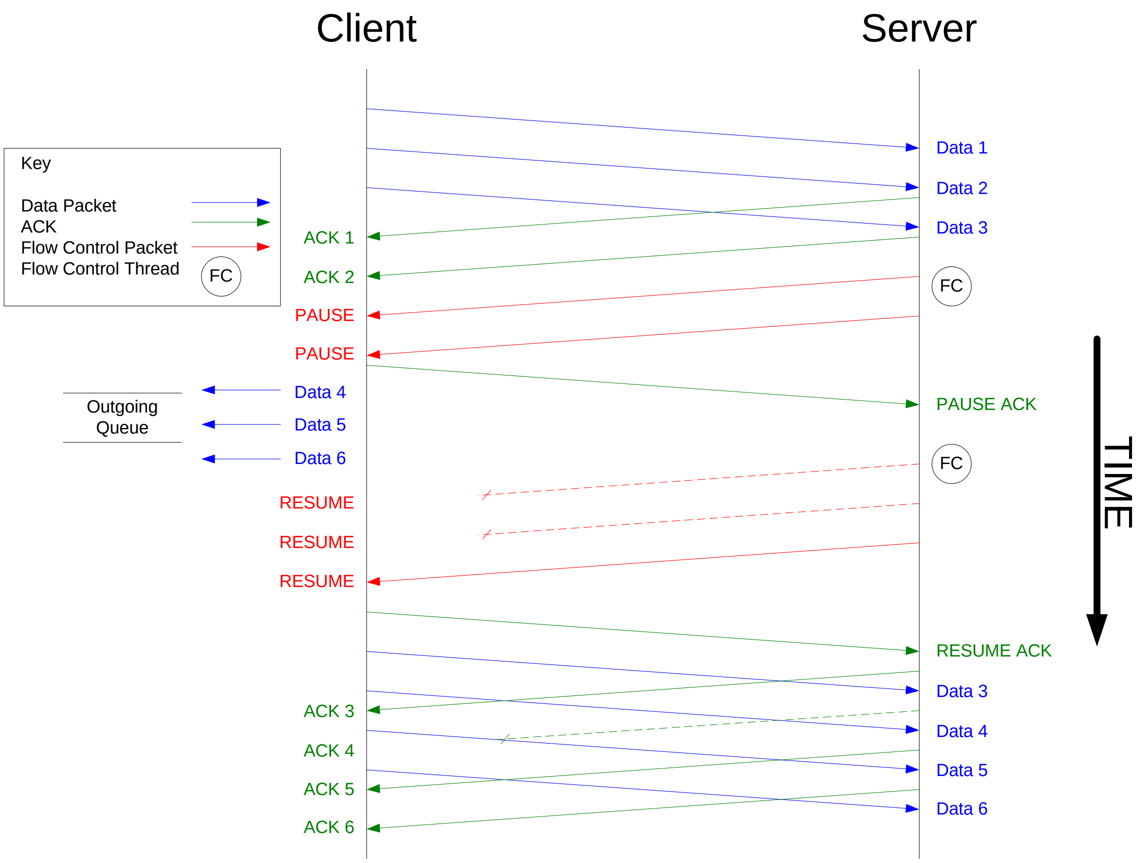


Figure 4. GridTcp Flow Control Time Chart

Phase 4 – Communication Performance Test

Phase 4 was a very short and simple test. The main purpose of this test was to see if an MPJ benchmark program would execute successfully with the new version of *GridTcp* as an *Ateam* program.

Unfortunately, there were older bugs from the previous version of *GridTcp* that I discovered during these tests that caused large time delays in the completion of *GridTcp.* In it's current state, *GridTcp* will run through most of the *AteamJGFPingPongBench* test successfully, but throws an OutOfMemoryError exception when large buffers are used. Please see the appendix for the test output. The bugs that had already existed in the version of *GridTcp* before I modified it were mainly thread synchronization issues. As is known throughout the field, this kind of bug can prove to be one of the most difficult to fix. In summary, *GridTcp* had a thread synchronization error that caused a deadlock when a *GridConnection*'s incoming que was being dequeued and enqueued at the same time. If *GridTcp*'s *receive* method received a null value from the dequeue operation it would sleep inside the *GridConnection*'s monitor and never be woken.

Skills Used and Developed

An important aspect of this project is to develop new skills and use old skills as I prepare for my career. The following is a list of some of the most important skills I have used and developed so far (in not particular order):

- Parallel programming
- The MPI API
- Multithreaded programming
- Multithreaded modeling
- Multithreaded debugging
- Knowledge of network stacks and TCP
- Serialization
- Inheritance: interfaces, abstract classes, method overriding, etc.
- Understanding of the Java language
- Understanding of the Java Virtual Machine
- Java packaging
- Java compilation
- Javadoc generation
- Linux shell scripting
- Linux security policies
- Technical writing
- Good commenting practices
- Code reading
- Modifying preexisting, large, complex software systems
- Java reflection
- Input/Output and Streams

Next Steps

The next phases of my project are as follows:

Phase 5: Enhancement/Implementation of File IOs in AgentTeamwork
I will create *FileInputStream* and *FileOutputStream* classes that wrap the *GridFile* classes created by Jumpei Miyauchi. Also, Jumpei and I will work together to implement *RandomAccessFile*, a file wrapper that will allow different nodes to access the same file in parallel albeit different partitions.

Phase 6: Enhancement of RandomAccessFile
Jumpei and I will work together to port/implement many of the file view features provided by MPI_IO.

Phase 7: File I/O Performance Evaluation and Conference Paper Submission
I will compare file-I/O performance between AgentTeamwork and NFS-based mpiJava in terms of the *FileInputStream* and *RandomAccessFile* classes. I will then work with Professor Fukuda to write and submit a paper to the either the PacRim ’07, GCA/PDPTA ’07, or other conference.

Files Created and Modified

The following is a list of files that have been either created or modified throughout the first half of the project:

File	Status	Changes/Use	Location
All AgentTeamwork source files	Old	Added package statements to almost every source file to restructure AgentTeamwork.	Medusa: /home/uwagent/agentteamwork-dev/
Ateam.java	New	For user-initiated snapshots. Added registerLocalVar() and retrieveLocalVar(). These methods allow local variables that are instantiated in main() to be serialized in a snapshot.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/
ATeamException.java	New	For reporting and describing errors that occur within AgentTeamwork	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/
AteamProg.java	New	Allows for the serialization of a static Ateam member that can be accessed in a user program's static main method	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/
backupToMedusa.sh	Dep.	Quickly backs up all files from local system to medusa	Koblab: /home/jawsh/tempAgentTeamworkBackup/
cleanMyNodeProcesses.sh	New	Kills orphan java process on Medusa's nodes. This orphan process sometimes prevent java sockets from binding.	Medusa: /home/uwagent/agentteamwork-dev/scripts/
Communicator.java	Old	Fixed simple bugs and added some documentation.	Medusa: /home/uwagent/agentteamwork-dev/MPJ/
CommunicatorTest.java	New	Tests the communication methods of MPJ as defined in phase 1 of my statement of work	Medusa: /home/uwagent/agentteamwork-dev/tests/
DiskVector.java	New	A class that extends java's AbstractList<E> and provides a list with the disk as a backing store. This class uses generics so it can be used for many general purposes.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp
genJavaDoc.sh	New	Creates consistent javadoc with complex command-line options	Medusa: /home/uwagent/agentteamwork-dev/scripts/
GridConnBackup.java	Dep.	A very simple class that includes a backup vector and connection ID's for serialization to disk.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridConnection.java	Old	Added a backup mechanism that writes old backup messages to disk when a specified threshold is reached. Also loads these persistent backups into memory when appropriate. Modified constructors and init() to allow for re-instantiation of a GridConnection with all of the memory management members included. Modified all methods to use DiskVector instead of GridConnBackup. Fixed some bugs.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/

GridConnMemChangeEvent.java	New	An event that is used by GridConnection to notify GridTcp (or other subscribers) of a change in memory	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridConnMemChangeListener.java	New	An interface that any subscriber to GridConnMemChangeEvent must implement.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridFlowControlThread.java	New	Simply continues to send PAUSE or RESUME packets at a specified interval until it is killed	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
All Grid Threads	Old	Each class that extends from the Thread class in GridTcp now sets its “thread name” in its constructor. This allows any GridTcp developer or user to easily determine which threads are running at any given time for debugging.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridPacket.java	Old	Added new packet types: data_ack, pause, resume, pause_ack, resume_ack	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridReceiveThread.java	Old	Added a temporary try/catch block to catch out of memory exceptions so that I can debug GridTcp’s memory issues.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridTcp.java	Old	Added a backup memory space threshold that defines how many bytes a GridTcp connection may store in memory before backing up old messages to persistent storage. Modified to use new functions modified in GridConnection. Modified the receive function to make incoming packet dequeing and sleeping an atomic operation if the packet returned is null. This is necessary because if an enqueue operation is occurring at the same time as a dequeue operation, there may be a readers-writers problem.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
GridTcpClientTest.java	New	Tests changes to GridTcp.	Medusa: /home/uwagent/agentteamwork-dev/tests/
GridTcpServerTest.java	New	Tests changes to GridTcp.	Medusa: /home/uwagent/agentteamwork-dev/tests/
GridUtil.java	Old	Added a simple method that retrieves the logon name of the current user. This is used when storing backup messages to disk. (SINCE REMOVED) Added a new function that prints all active threads currently running within the JVM for debugging purposes.	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/GridTcp/
IRecvThread.java	Old	Reformatting, comments, and javadoc	Medusa: /home/uwagent/agentteamwork-dev/MPJ/
ISendThread.java	Old	Reformatting, comments, and javadoc	Medusa: /home/uwagent/agentteamwork-dev/MPJ/
javadoc	New	Javadoc for all of AgentTeamwork	Medusa: /home/uwagent/agentteamwork-dev/doc/

JGF tests	Old	Eliminated the use of the jgfutil package so that it would run correctly. Also ported PingPongBench and AllgatherBench to Ateam programs.	Medusa: /home/uwagent/agentteamwork-dev/MPJ/JGF/
JGFMaster.sh, JGFSlave.sh	New	Runs JGF tests.	Medusa: /home/uwagent/agentteamwork-dev/JGF/
Misc. Script Files	New	For backup, javadoc generation, and compilation of AgentTeamwork	Medusa: /home/uwagent/agentteamwork-dev/scripts
Misc. Test Files	New	For testing serialization and package compilation issues.	Medusa: /home/uwagent/agentteamwork-dev/tests/
Mpjrun.java	Old	Reformatting, comments, and javadoc. Also changed parameters parsing to look for new versions of parameters. (i.e. –slave instead of –amslave)	Medusa: /home/uwagent/agentteamwork-dev/MPJ/
Request.java	Old	Reformatting, comments, and javadoc. Fixed Illegal Monitor State bug.	Medusa: /home/uwagent/agentteamwork-dev/MPJ/
runCommunicatorTest.sh	New	Launches CommunicatorTest	Medusa: /home/uwagent/agentteamwork-dev/tests/
ServerSocket.java	New	Wraps GridServerSocket.java	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/
Socket.java	New	Wraps GridSocket.java	Medusa: /home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/
UPWTest.java	New	Now tests Ateam by extending the AteamProg class.	Medusa: /home/uwagent/agentteamwork-dev/tests/
UserProgWrapper.java	Old	Added support for new and old versions of AgentTeamwork (i.e.-partitioned and non-partitioned). Added support for AteamProg class as well as instantiation of GridTcp for Ateam programs. Added support for AteamProg class as well as instantiation of GridTcp for Ateam programs. Also added a new parameter for main that accepts a port number to use when instantiating GridTcp.	Medusa: //home/uwagent/agentteamwork-dev/AgentTeamwork/Ateam/

Table 1. Files created or modified in Phases 1 - 4

Future Project Recommendations

While working on AgentTeamwork I have compiled a short list of some project recommendations that future contributers might implement. They are:

- Reformatting and commenting of Communicator.java
- Creation of an MPJException class
- Argument checking and informative exception details for MPJ communication methods. (e.g. – If the user calls Reduce() and receiveBuffer.length < recvCount + recvOffset, an MPJException is thrown in which this problem is explained).

- I recommend that at some point, large packets be fragmented. This will alleviate many of the memory issues that occur within GridTcp. Then, the memory threshold for GridTcp should automatically adjust to be the closest multiple of this packet size. I don't think it would be too difficult to implement.
- It may be possible (further research would be necessary) to provide an additional memory safeguard in GridTcp that would automatically kick in packet backup to disk when the available memory nearly reaches 0.
- Much more advanced and deeper testing of GridTcp's new flow control feature

Appendix A – JGFPingPongBench Results After Phase 1

JGFPingPongBench (2 nodes) Results

```
[jawsh@medusa JGF]$ ./JGFPingPongBenchM.sh
Running test as the master node..
Master accepted connection from mnode14
Master trying to read slave rank
Master got slave 1
Java Grande Forum MPJ Benchmark Suite - Version 1.0 - Section 1
Executing on 2 processes

Section1:PingPong:Double          83207.11     (bytes/s)    Array Size = 4
Section1:PingPong:Double          146332.38     (bytes/s)    Array Size = 7
Section1:PingPong:Double          252203.0     (bytes/s)    Array Size = 12
Section1:PingPong:Double          333375.16     (bytes/s)    Array Size = 21
Section1:PingPong:Double          498347.03     (bytes/s)    Array Size = 37
Section1:PingPong:Double          693280.3     (bytes/s)    Array Size = 66
Section1:PingPong:Double          816999.06     (bytes/s)    Array Size = 116
Section1:PingPong:Double          988616.2     (bytes/s)    Array Size = 203
Section1:PingPong:Double          1335713.1     (bytes/s)    Array Size = 357
Section1:PingPong:Double          1725792.4     (bytes/s)    Array Size = 626
Section1:PingPong:Double          1999292.2     (bytes/s)    Array Size = 1098
Section1:PingPong:Double          2238461.0     (bytes/s)    Array Size = 1926
Section1:PingPong:Double          2413363.2     (bytes/s)    Array Size = 3377
Section1:PingPong:Double          2518161.8     (bytes/s)    Array Size = 5921
Section1:PingPong:Double          2889575.2     (bytes/s)    Array Size = 10383
Section1:PingPong:Double          3358602.0     (bytes/s)    Array Size = 18205
Section1:PingPong:Double          3627869.5     (bytes/s)    Array Size = 31921
Section1:PingPong:Double          3869642.8     (bytes/s)    Array Size = 55970
Section1:PingPong:Double          3978632.0     (bytes/s)    Array Size = 98137
Section1:PingPong:Double          4068196.5     (bytes/s)    Array Size = 172072
Section1:PingPong:Double          3946112.0     (bytes/s)    Array Size = 301708
Section1:PingPong:Double          4163896.2     (bytes/s)    Array Size = 529010
Section1:PingPong:Double          4154209.0     (bytes/s)    Array Size = 927557
Section1:PingPong:Double          4179198.0     (bytes/s)    Array Size = 1626361
Section1:PingPong:Double          4176685.5     (bytes/s)    Array Size = 2851632
Section1:PingPong:Object         6187.311     (objects/s)    Array Size = 4
Section1:PingPong:Object         10753.681     (objects/s)    Array Size = 7
Section1:PingPong:Object         15419.026     (objects/s)    Array Size = 12
Section1:PingPong:Object         20640.951     (objects/s)    Array Size = 21
Section1:PingPong:Object         28508.652     (objects/s)    Array Size = 37
Section1:PingPong:Object         34111.797     (objects/s)    Array Size = 66
Section1:PingPong:Object         39991.246     (objects/s)    Array Size = 116
Section1:PingPong:Object         48919.69     (objects/s)    Array Size = 203
Section1:PingPong:Object         56271.53     (objects/s)    Array Size = 357
Section1:PingPong:Object         61336.14     (objects/s)    Array Size = 626
Section1:PingPong:Object         64845.26     (objects/s)    Array Size = 1098
Section1:PingPong:Object         67000.41     (objects/s)    Array Size = 1926
Section1:PingPong:Object         59539.395     (objects/s)    Array Size = 3377
Section1:PingPong:Object         52296.992     (objects/s)    Array Size = 5921
Section1:PingPong:Object         47755.086     (objects/s)    Array Size = 10383
```

```
Section1:PingPong:Object         46578.715    (objects/s)    Array Size = 18205
Section1:PingPong:Object         45867.625    (objects/s)    Array Size = 31921
Section1:PingPong:Object         46402.402    (objects/s)    Array Size = 55970
Section1:PingPong:Object         40905.33     (objects/s)    Array Size = 98137
Section1:PingPong:Object         51487.73     (objects/s)    Array Size = 172072
Section1:PingPong:Object         50796.867    (objects/s)    Array Size = 301708
Section1:PingPong:Object         48337.902    (objects/s)    Array Size = 529010
Section1:PingPong:Object         45955.062    (objects/s)    Array Size = 927557
Section1:PingPong:Object         44691.297    (objects/s)    Array Size = 1626361
Section1:PingPong:Object         48975.234    (objects/s)    Array Size = 2851632
MPJRUN_READERTHREAD_EXIT
```

JGFGatherBench (2 nodes) Results

```
[jawsh@medusa JGF]$ ./JGFMaster.sh JGFGatherBench
Running test as the master node..
Master accepted connection from mnode14
Master trying to read slave rank
Master got slave 1
Java Grande Forum MPJ Benchmark Suite - Version 1.0 - Section 1
Executing on 2 processes

Section1:Gather:Double           117553.36    (bytes/s)      Array Size = 4
Section1:Gather:Double           311123.78    (bytes/s)      Array Size = 7
Section1:Gather:Double           666044.44    (bytes/s)      Array Size = 12
Section1:Gather:Double           269537.0     (bytes/s)      Array Size = 21
Section1:Gather:Double           715078.75    (bytes/s)      Array Size = 37
Section1:Gather:Double           2072781.1    (bytes/s)      Array Size = 66
Section1:Gather:Double           1839607.0    (bytes/s)      Array Size = 116
Section1:Gather:Double           1987719.8    (bytes/s)      Array Size = 203
Section1:Gather:Double           3118889.8    (bytes/s)      Array Size = 357
Section1:Gather:Double           3510056.2    (bytes/s)      Array Size = 626
Section1:Gather:Double           2622013.0    (bytes/s)      Array Size = 1098
Section1:Gather:Double           4705224.0    (bytes/s)      Array Size = 1926
Section1:Gather:Double           4990868.5    (bytes/s)      Array Size = 3377
Section1:Gather:Double           5192403.0    (bytes/s)      Array Size = 5921
Section1:Gather:Double           5792531.5    (bytes/s)      Array Size = 10383
Section1:Gather:Double           6576213.0    (bytes/s)      Array Size = 18205
Section1:Gather:Double           7094325.5    (bytes/s)      Array Size = 31921
Section1:Gather:Double           7452962.5    (bytes/s)      Array Size = 55970
Section1:Gather:Double           7635612.0    (bytes/s)      Array Size = 98137
Section1:Gather:Double           7756723.5    (bytes/s)      Array Size = 172072
Section1:Gather:Double           7679949.0    (bytes/s)      Array Size = 301708
Section1:Gather:Double           7894291.0    (bytes/s)      Array Size = 529010
Section1:Gather:Double           7897252.5    (bytes/s)      Array Size = 927557
Section1:Gather:Double           7902749.0    (bytes/s)      Array Size = 1626361
Section1:Gather:Double           7903362.5    (bytes/s)      Array Size = 2851632
Section1:Gather:Object           13875.927    (objects/s)    Array Size = 4
Section1:Gather:Object           24349.895    (objects/s)    Array Size = 7
Section1:Gather:Object           8909.189     (objects/s)    Array Size = 12
Section1:Gather:Object           28181.178    (objects/s)    Array Size = 21
Section1:Gather:Object           72765.336    (objects/s)    Array Size = 37
Section1:Gather:Object           85820.95     (objects/s)    Array Size = 66
Section1:Gather:Object           91725.09     (objects/s)    Array Size = 116
Section1:Gather:Object           114695.91    (objects/s)    Array Size = 203
Section1:Gather:Object           133492.06    (objects/s)    Array Size = 357
Section1:Gather:Object           132060.98    (objects/s)    Array Size = 626
Section1:Gather:Object           153035.53    (objects/s)    Array Size = 1098
Section1:Gather:Object           160932.19    (objects/s)    Array Size = 1926
Section1:Gather:Object           162548.1     (objects/s)    Array Size = 3377
Section1:Gather:Object           161381.53    (objects/s)    Array Size = 5921
Section1:Gather:Object           157935.11    (objects/s)    Array Size = 10383
Section1:Gather:Object           162510.64    (objects/s)    Array Size = 18205
Section1:Gather:Object           160659.33    (objects/s)    Array Size = 31921
Section1:Gather:Object           159316.84    (objects/s)    Array Size = 55970
Section1:Gather:Object           148523.64    (objects/s)    Array Size = 98137
Section1:Gather:Object           126302.96    (objects/s)    Array Size = 172072
Section1:Gather:Object           115342.83    (objects/s)    Array Size = 301708
```

```
Section1:Gather:Object          121131.15     (objects/s)    Array Size = 529010
Section1:Gather:Object          118280.67     (objects/s)    Array Size = 927557
Section1:Gather:Object          98501.664     (objects/s)    Array Size = 1626361
Section1:Gather:Object          108741.305     (objects/s)    Array Size = 2851632
MPJRUN_READERTHREAD_EXIT
```

JGFGatherBench (4 nodes) Results

```
[jawsh@medusa JGF]$ ./JGFMaster.sh JGFGatherBench 4
Running test as the master node..
Master accepted connection from mnode14
Master trying to read slave rank
Master got slave 1
Master accepted connection from mnode15
Master trying to read slave rank
Master got slave 2
Master accepted connection from mnode16
Master trying to read slave rank
Master got slave 3
Java Grande Forum MPJ Benchmark Suite - Version 1.0 - Section 1
Executing on 4 processes

Section1:Gather:Double          118096.18     (bytes/s)     Array Size = 4
Section1:Gather:Double          243305.22     (bytes/s)     Array Size = 7
Section1:Gather:Double          165251.52     (bytes/s)     Array Size = 12
Section1:Gather:Double          347451.66     (bytes/s)     Array Size = 21
Section1:Gather:Double          595562.3     (bytes/s)     Array Size = 37
Section1:Gather:Double          1214481.5     (bytes/s)     Array Size = 66
Section1:Gather:Double          1433020.9     (bytes/s)     Array Size = 116
Section1:Gather:Double          1226722.8     (bytes/s)     Array Size = 203
Section1:Gather:Double          1779596.2     (bytes/s)     Array Size = 357
Section1:Gather:Double          1914933.5     (bytes/s)     Array Size = 626
Section1:Gather:Double          1514023.9     (bytes/s)     Array Size = 1098
Section1:Gather:Double          1985502.0     (bytes/s)     Array Size = 1926
Section1:Gather:Double          1021957.3     (bytes/s)     Array Size = 3377
Section1:Gather:Double          1782479.5     (bytes/s)     Array Size = 5921
Section1:Gather:Double          1386385.8     (bytes/s)     Array Size = 10383
Section1:Gather:Double          1801151.8     (bytes/s)     Array Size = 18205
Section1:Gather:Double          1170490.0     (bytes/s)     Array Size = 31921
Section1:Gather:Double          2097236.5     (bytes/s)     Array Size = 55970
Section1:Gather:Double          1842948.4     (bytes/s)     Array Size = 98137
Section1:Gather:Double          2228483.5     (bytes/s)     Array Size = 172072
Section1:Gather:Double          2042881.1     (bytes/s)     Array Size = 301708
Section1:Gather:Double          2272562.8     (bytes/s)     Array Size = 529010
Section1:Gather:Double          2453246.0     (bytes/s)     Array Size = 927557
Section1:Gather:Double          2528841.2     (bytes/s)     Array Size = 1626361
Section1:Gather:Double          2564129.0     (bytes/s)     Array Size = 2851632
Section1:Gather:Object          6839.4907     (objects/s)    Array Size = 4
Section1:Gather:Object          11314.917     (objects/s)    Array Size = 7
Section1:Gather:Object          15542.135     (objects/s)    Array Size = 12
Section1:Gather:Object          27213.795     (objects/s)    Array Size = 21
Section1:Gather:Object          35794.047     (objects/s)    Array Size = 37
Section1:Gather:Object          41825.016     (objects/s)    Array Size = 66
Section1:Gather:Object          45528.555     (objects/s)    Array Size = 116
Section1:Gather:Object          54836.64     (objects/s)    Array Size = 203
Section1:Gather:Object          59466.125     (objects/s)    Array Size = 357
Section1:Gather:Object          58890.582     (objects/s)    Array Size = 626
Section1:Gather:Object          51947.516     (objects/s)    Array Size = 1098
Section1:Gather:Object          43178.562     (objects/s)    Array Size = 1926
Section1:Gather:Object          42033.938     (objects/s)    Array Size = 3377
Section1:Gather:Object          59493.523     (objects/s)    Array Size = 5921
Section1:Gather:Object          62220.227     (objects/s)    Array Size = 10383
Section1:Gather:Object          57851.043     (objects/s)    Array Size = 18205
Section1:Gather:Object          59429.37     (objects/s)    Array Size = 31921
Section1:Gather:Object          62483.953     (objects/s)    Array Size = 55970
Section1:Gather:Object          59571.742     (objects/s)    Array Size = 98137
Section1:Gather:Object          56177.605     (objects/s)    Array Size = 172072
```

```
Section1:Gather:Object             52466.395      (objects/s)    Array Size = 301708
Section1:Gather:Object             50729.766      (objects/s)    Array Size = 529010
Section1:Gather:Object             49981.516      (objects/s)    Array Size = 927557
Section1:Gather:Object             47647.76      (objects/s)    Array Size = 1626361
Section1:Gather:Object             49024.066      (objects/s)    Array Size = 2851632
MPJRUN_READERTHREAD_EXIT
```

JGFGatherBench (8 nodes) Results

```
[jawsh@medusa JGF]$ ./JGFMaster.sh JGFGatherBench 8
Running test as the master node..
Master accepted connection from mnode14
Master trying to read slave rank
Master got slave 1
Master accepted connection from mnode15
Master trying to read slave rank
Master got slave 2
Master accepted connection from mnode16
Master trying to read slave rank
Master got slave 3
Master accepted connection from mnode17
Master trying to read slave rank
Master got slave 4
Master accepted connection from mnode18
Master trying to read slave rank
Master got slave 5
Master accepted connection from mnode19
Master trying to read slave rank
Master got slave 6
Master accepted connection from mnode20
Master trying to read slave rank
Master got slave 7
Java Grande Forum MPJ Benchmark Suite - Version 1.0 - Section 1
Executing on 8 processes

Section1:Gather:Double             90068.375      (bytes/s)      Array Size = 4
Section1:Gather:Double             110184.22      (bytes/s)      Array Size = 7
Section1:Gather:Double             124140.805      (bytes/s)      Array Size = 12
Section1:Gather:Double             415160.2      (bytes/s)      Array Size = 21
Section1:Gather:Double             381863.3      (bytes/s)      Array Size = 37
Section1:Gather:Double             597840.5      (bytes/s)      Array Size = 66
Section1:Gather:Double             709489.1      (bytes/s)      Array Size = 116
Section1:Gather:Double             567956.3      (bytes/s)      Array Size = 203
Section1:Gather:Double             939461.6      (bytes/s)      Array Size = 357
Section1:Gather:Double             186737.75      (bytes/s)      Array Size = 626
Section1:Gather:Double             153171.03      (bytes/s)      Array Size = 1098
Section1:Gather:Double             230507.72      (bytes/s)      Array Size = 1926
Section1:Gather:Double             148007.53      (bytes/s)      Array Size = 3377
Section1:Gather:Double             282767.66      (bytes/s)      Array Size = 5921
Section1:Gather:Double             309813.84      (bytes/s)      Array Size = 10383
Section1:Gather:Double             500400.5      (bytes/s)      Array Size = 18205
Section1:Gather:Double             577960.0      (bytes/s)      Array Size = 31921
Section1:Gather:Double             748840.8      (bytes/s)      Array Size = 55970
Section1:Gather:Double             766976.2      (bytes/s)      Array Size = 98137
Section1:Gather:Double             885756.3      (bytes/s)      Array Size = 172072
Section1:Gather:Double             954961.0      (bytes/s)      Array Size = 301708
Section1:Gather:Double             1024158.75      (bytes/s)      Array Size = 529010
Section1:Gather:Double             1061126.2      (bytes/s)      Array Size = 927557
Section1:Gather:Double             1083789.1      (bytes/s)      Array Size = 1626361
Section1:Gather:Double             1106624.1      (bytes/s)      Array Size = 2851632
Section1:Gather:Object             2466.9126      (objects/s)    Array Size = 4
Section1:Gather:Object             5086.8447      (objects/s)    Array Size = 7
Section1:Gather:Object             7473.316      (objects/s)    Array Size = 12
Section1:Gather:Object             9353.632      (objects/s)    Array Size = 21
Section1:Gather:Object             16525.133      (objects/s)    Array Size = 37
Section1:Gather:Object             22362.146      (objects/s)    Array Size = 66
```

```
Section1:Gather:Object          23542.562    (objects/s)   Array Size = 116
Section1:Gather:Object          26687.893    (objects/s)   Array Size = 203
Section1:Gather:Object          7295.019    (objects/s)   Array Size = 357
Section1:Gather:Object          8786.446    (objects/s)   Array Size = 626
Section1:Gather:Object          9501.994    (objects/s)   Array Size = 1098
Section1:Gather:Object          10912.182    (objects/s)   Array Size = 1926
Section1:Gather:Object          15238.525    (objects/s)   Array Size = 3377
Section1:Gather:Object          18621.326    (objects/s)   Array Size = 5921
Section1:Gather:Object          23887.842    (objects/s)   Array Size = 10383
Section1:Gather:Object          26324.447    (objects/s)   Array Size = 18205
Section1:Gather:Object          22733.73    (objects/s)   Array Size = 31921
Section1:Gather:Object          22161.947    (objects/s)   Array Size = 55970
Section1:Gather:Object          23164.64    (objects/s)   Array Size = 98137
Section1:Gather:Object          23616.799    (objects/s)   Array Size = 172072
Section1:Gather:Object          22535.703    (objects/s)   Array Size = 301708
Section1:Gather:Object          23192.021    (objects/s)   Array Size = 529010
Section1:Gather:Object          23427.297    (objects/s)   Array Size = 927557
Section1:Gather:Object          23066.332    (objects/s)   Array Size = 1626361
Section1:Gather:Object          23336.35    (objects/s)   Array Size = 2851632
MPJRUN_READERTHREAD_EXIT
```

Appendix B – Example Ateam User Program

```
import AgentTeamwork.Ateam.*;

public class UPWTest extends AteamProg {

        private int phase;
        public int[] intBuf;
        public int x;

        // blank const for Ateam
        public UPWTest( Object o ) { }

        public UPWTest( ) {
                phase = 0;
                x = 0;
        }

        private void userRecovery( ) {
                try {
                        phase = ateam.getSnapshotId( );
                } catch ( Exception e ) {
                        e.printStackTrace( );
                        System.exit( 1 );
                }
        }

        // does nothing but change the values of an array, and take a
        // snapshot after each change
        private void compute( ) {
                try {
                        intBuf = new int[100];
                        for( int i = phase; i < 100; i++ ) {
                                for( int j = 0; j < intBuf.length; j++ ) {
                                        intBuf[ j ] = i;
                                        x++;
                                }
                                System.out.println( "Taking a snapshot" );
                                ateam.takeSnapshot( i );
                        }
                        System.out.println( "Finished execution!" );
                } catch ( Exception e ) {
                        e.printStackTrace( );
                        System.exit( 1 );
                }
        }

        public static void main( String[] args ) {
                 UPWTest program = null;
                if ( ateam.isResumed( ) ) {
                    program = (UPWTest) ateam.retrieveLocalVar( "program" );
                    program.userRecovery( );
                } else {
                    program = new UPWTest( );
                    ateam.registerLocalVar( "program", program );
                }
                program.compute( );
        }
}
```

Appendix C – Explanation of Source Development Tree

```
+-----------------------------------------------------------------------------+
| AGENT TEAMWORK DEVELOPMENT TREE                                             |
|    A detailed explanation                                                   |
|                                                                             |
|    Author  - Joshua Phillips (jawsh@u.washington.edu)                       |
|    Version - 11/02/2006                                                     |
+-----------------------------------------------------------------------------+

This is the root directory of the AgentTeamwork Development tree.  All source
code should be compiled from this directory. This will allow the java compiler
to easily find all imported packages.


*---- DIRECTORY STRUCTURE ----*

Note that MPJ and UWAgent packages reside outside of the AgentTeamwork package
because they can be used as stand-alone packages.

The structure of this development tree is as follows:

agentteamwork-dev
   |
   +----AgentTeamwork
   |         |
   |         +----Agents
   |         |
   |         +----Ateam
   |               |
   |               +----GridFile
   |               |
   |               +----GridTcp
   |
   +----MPJ
   |     |
   |     +----JGF
   |
   +----UWAgent
   |
   +----jars
   |
   +----tests
   |
   +----scripts

Each package directory (e.g. AgentTeamwork/Ateam/GridFile is organized in the following
manner:

        * package source files (NOTE: Nothing but SOURCE files and DIRECTORIES should be
               placed in a package directory)
        * "other" directory - contains scripts, backups, and documentation



*---- PACKAGES ----*

AgentTeamwork.Agents:
        Contains all agents: Commander, Sentinel, Bookkeeper, etc.
AgentTeamwork.Ateam.GridFile:
        Contains the GridFile classes for an error recoverable file.
```

```
AgentTeamwork.Ateam.GridTcp:
        Contains the GridTcp classes for an error recoverable TCP connection.
MPJ:
        Contains all classes for MPJ (Message Passing Java).
UWAgent:
        Contains all classes for UWAgent, the mobile agents that AgentTeamwork relies on.


*---- OTHER ----*

Please note that all tests should be placed in the "tests" folder, NOT in source
directories.
The "jars" folder contains all of the compiled and archived packages.
The "scripts" folder can contain backup scripts and compiling scripts.


*---- SCRIPTS ----*
Within the scripts folder are maintenance scripts for backup, compilation, archiving, and
cleaning of the AgentTeamwork development tree. There are scripts for compiling specific
packages and for the entire dev tree:

        - compileAndPack********.sh
                        Compiles that specific package, places all class files into a jar
                        file the /agentteamworrk-dev/jars directory and cleans the package
                        directory by removing class files.
        - compileAndPackAllPackages.sh
                        Compiles, packs, and cleans every package in the AgentTeamwork dev-
                        tree.  All package jar files are placed in the /agentteamwork-
                        dev/jars directory.
```

References

1. Sun’s Javadoc Guide
   [http://java.sun.com/j2se/javadoc/writingdoccomments]