CSS497 Winter 2006
Enhancing AgentTeamwork for Inter-Cluster Deployment of Agents

Intermediate Report: Fri 3/24/06

Summary

Dates: 1/1/06 – 3/24/06

Summary of Work:

- Ported AgentTeamwork's agents to the Java-socket-based version of UWAgent mobile-agent execution platform that allows agents to migrate over a cluster gateway.
- Imported SensorAgent remote resource availability enhancements developed by Jun Morisaki.
- Enhanced AgentTeamwork's sentinel, commander, and resource agents, so that a collection of sentinel agents deploy a parallel application over two or more cluster systems.
- Imported AgentTeamwork GUI and file transfer component developed by Jumpei Miyauchi.

Source Code

This section shows some of the new and changed source code.

1. Porting of AgentTeamwork's agents to the Java-socket-based version of UWAgent.

Removed RMI code from all agents.
Modified agent code to use new base AgentTeamwork methods.
Removed obsolete code from agents.
Removed registerDomainAccess method from AgentUtil.java.

2. Imported SensorAgent remote resource availability enhancements developed by Jun Morisaki.

Extensive rewrite of SensorAgent.java by Jun.
Minor changes to ResourceAgent.java.

3. Imported AgentTeamwork GUI and file transfer component developed by Jumpei Miyauchi.

New files added:

- GridFile.java
- GridFileInputStream.java
- GridFileOutputStream.java
- UserProgWrapper.java
- SubmitGUI.java

Major changes to **CommanderAgent.java** from Jumpei:

- receiveStdout() rewritten.
- Changes to ReceiveStdin()
- sendInFiles() rewritten.

Major changes to **SentinelAgent.java** from Jumpei:

- New file-i/o member variables.
- New code to initArgs() to initialize file-i/o member variables.
- New code to funcMethod() to retrieve job output.
- Rewrite of receiveInFile().
- Rewrite of receiveStdin().
- Rewrite of receiveUserData().
- Rewrite of receiveOutFiles().

Major changes to **AgentUtil.java** from Jumpei:

- Minor changes to sendStdin().

4. Enhancements to AgentTeamwork's agents for deploying a parallel application over two or more cluster systems.

Major changes to CommanderAgent.java:

// New member variables

```
private List clusters = new ArrayList(); // Remote cluster names and nodes
private int numClusterNodes = 0;        // Total sum of all cluster nodes.
private List clustersExtra = new ArrayList(); // Remote cluster names and nodes
private int numExtraClusterNodes = 0;        // Total sum of all extra cluster nodes.
```

// New launch parameters

```
//            <li> {CL_cluster_gateway_ipname{[_cluster_node_ipname]}}
//                  Specifies a remote cluster to use as well.
//                  The cluster gateway name comes first, followed by a
//                  list of machine nodes within that cluster.
//                  If it is not given, a resource agent is responsible
//                  to provide the commander with such a list.
        if ( args[i].startsWith( "CL_" ) ) {
            // CL option: a remote cluster and list of cluster nodes
            String[] sA = null;
            sA = args[i].split( "_" );
            if ( sA == null || sA.length < 3 ) {
                // no ip names
                usage( "CL option requires cluster name, and 1 or more cluster nodes" );
            }
            // Add it to list of clusters, for later processing.
            clusters.add( args[i] );
            numClusterNodes += (sA.length - 2);
        }
//            <li> {ECL_cluster_gateway_ipname{[_cluster_node_ipname]}}
//                  Specifies extra remote clusters to use.
//                  The cluster gateway name comes first, followed by a
//                  list of machine nodes within that cluster.
//                  If it is not given, a resource agent is responsible
//                  to provide the commander with such a list.
        else if ( args[i].startsWith( "ECL_" ) ) {
            // ECL option: an extra remote cluster and list of cluster nodes
            String[] sA = null;
            sA = args[i].split( "_" );
            if ( sA == null || sA.length < 3 ) {
                // no ip names
                usage( "ECL option requires cluster name, and 1 or more cluster nodes" );
            }
            // Add it to list of clusters, for later processing.
            clustersExtra.add( args[i] );
            numExtraClusterNodes += (sA.length - 2);
        }
```

// New version of spawnSentinel()

```
private void spawnSentinel( ) {
    // memorize the size of nArgs.
    int nArgsLength = ( nArgs != null ) ? nArgs.length : 0;

    //
    // construct a list of arguments passed to a sentinel
    //

    //ECH: New params for remote clusters
    String[] sentinelArgs = new String[ 2 + clusters.size() * 3 +
                                    numClusterNodes +
                                    1 + clustersExtra.size() * 3 +
                                    numExtraClusterNodes +
                                    sArgs.length - 1 +
                                    userProgArray.length +
                                    nArgsLength ];

    //
    //ECH: Process the remote clusters
    //

    //ECH: Set total# of computing nodes (local + remote)
    sentinelArgs[0] = String.valueOf( sArgs.length - 1 + numClusterNodes );
    //ECH: Set number of remote clusters
    sentinelArgs[1] = String.valueOf( clusters.size() );
    int index = 2;
    int indexClusterDetails = index + clusters.size() * 2;
    //ECH: Populate list of cluster names, counts, and cluster nodes
    for (int i=0; i < clusters.size(); i++) {
        String[] args = clusters[i].split( "_" );
        // Cluster name
        sentinelArgs[index + i] = args[1];          // Ignore leading "CL_" tag
        // Cluster node count
        sentinelArgs[index + i * 2] = String.valueOf( args.length - 2 );
        // Cluster name, again. BUGBUG: Should this be something different?
        sentinelArgs[indexClusterDetails] = args[1];
        // List of computing nodes within the cluster
        for (int j=0; j < args.length - 2; j++) {
            sentinelArgs[indexClusterDetails + 1 + j] = args[j + 2];
        }
        indexClusterDetails += (args.length - 1);
    }
    index = indexClusterDetails;   // The local stuff gets appended after us

    //
    // Now process the local nodes
    //

    // sArgs.length = S + #ipnames, which means
    // sArgs.length - 1 = #sentinels
    // overwrite sArgs[0] with #sentinels
    sArgs[index] = String.valueOf( sArgs.length - 1 );

    for ( int j = 0; j < sArgs.length; j++ ) {
        // #sentinels followed by ipnames
        sentinelArgs[index++] = sArgs[j];
    }

    //
    // Process the user-program name/arguments
    //

    for (int j = 1; j < userProgArray.length; j++) {
        // user prog and args
        // userProgArray[0]=="U", The actual prog/args start from [1]
        sentinelArgs[index++] = userProgArray[j];
    }
```

```
        // nArgs[0] = "_?$end_of_user$?_"
        // need this delimitor between user program arguments and extra
        // ip names
        sentinelArgs[index++] = "_?$end_of_user$?_";

        //
        //ECH: Process the extra-clusters
        //TODO: Need to implement this!
        //
        sentinelArgs[index++] = String.valueOf( 0 );

        //
        // Finally process the extra local nodes
        //
        for (int j = 1; j < nArgsLength; j++) {
            // extra ipnames if a sentinel crashes
            sentinelArgs[index++] = nArgs[j];
        }
        // for debugging
        for (int i = 0; i < sentinelArgs.length; i++)
            printErr( "sentinelArgs[" + i + "] = " + sentinelArgs[i] );

        // spawn a root sentinel agent that will further spawn its children
        // using sentinelArgs
        if ( userProgName != null ) {

            // prepare a uwagent that will point to a new agent
            UWAgent uwa = null;

            // prepare all classes accessed by and thus carried with a sentinel
            String[] classNames = null;
            if ( userClassNames != null ) {
                // userClassNames[0] == "C" which is now replaced with
                // userProgName
                userClassNames[0] = userProgName;
                classNames = userClassNames;
            }
            else {
                // no additional classes other than userProgName
                classNames = agentutil.attachClassName( userProgName );
            }
            // we need to attach AgentUtil to them, too.
            classNames = agentutil.attachClassName( classNames, "AgentUtil" );

            // spawn a child as a root sentinel agent
            uwa = spawnChild( "SentinelAgent", sentinelArgs, sArgs[1],
                              classNames );
            if ( uwa != null ) {
                // upon a success, memorize the root sentinel agent's id
                int childId = uwa.getAgentId( ); // retrieve my new child id
                printErr( "spawn SentinelAgent (id = " + childId + ")" );
                rootSentinelId = childId;
            }
        }
        printErr( "spawnSentinel: completed" );
    }
```

Major changes to AgentUtil.java:

// New calculations for remote clusters:

```
    // Returns 0 for left side (cluster gateways/node), or 1 for right side (local nodes).
    public int calculateTreeSide( int agentId ) {
        int maxChildren = myAgent.getMaxChildren( );
        int idRoot = Math.pow( maxChildren, 3 );

        while ( agentId > idRoot + 1 ) {
            agentId = (int)(agentId / maxChildren);
        }
    agentId -= idRoot;
    if ( agentId < 0 ) {
        printErr( "AgentUtil.calculateTreeSide: agentId < 0!" );
    }
    return agentId;
}

// Calculate the inner layer, that is, the layer within a cluster.
public int calculateInnerLayer( int agentId, int rootAgentId ) {
    int maxChildren = myAgent.getMaxChildren( );
    int layer = ( int )(( Math.log( agentId ) / Math.log( maxChildren ) )
        / rootAgentId );
}

// Calculate the agent's offset within its rank.
public int calculateOffsetFromLeftMostAgent( int agentId, rootAgentId ) {
    int maxChildren = myAgent.getMaxChildren( );
    return agentId - rootAgentId * Math.pow( maxChildren,
        calculateInnerLayer( agentId, rootAgentId ) );
}

// Calculate the ancestor in charge of the cluster gateway for this agent.
public int calculateClusterGatewayNodeId( int agentId ) {
    int maxChildren = myAgent.getMaxChildren( );
    int idRoot = Math.pow( maxChildren, 3 );

    // Build ancestor-id array
    int sizeArray = (int)(( Math.log( agentId ) / Math.log( maxChildren ) );
    int[] idArray = new int[sizeArray];
    int i = 0;
    while ( agentID > idRoot ) {
        agentId = (int)(agentId / maxChildren);
        idArray[i] = agentId;
        i++;
    }

    // Find right-most 4-divisible non-8 agent id, and return it
    while ( i > 0 ) {
        i--;
        if ((idArray[i] % maxChildren) == 0)
            return idArray[i];
    }

    // Didn't find ancestor,  Return error.
    printErr( "AgentUtil.calculateClusterGatewayNodeId: Didn't find 4-divisible non-8 agent ID!" );
    return -1;
}

// Calculates 0-based cluster# from culter gateway id.
public int calculateClusterNumber( int clusterGatewayId ) {
    int maxChildren = myAgent.getMaxChildren( );
    int idRoot = Math.pow( maxChildren, 3 );

    // Build remainder array
    int sizeArray = (int)(( Math.log( agentId ) / Math.log( maxChildren ) );
    int[] idArray = new int[sizeArray];
    int i = 0;
    int id = clusterGatewayId;
    int remainder;
    while ( id > idRoot ) {
        remainder = id % maxChildren;
        id = (int)(id / maxChildren);
        idArray[i] = remainder;
        i++;
    }

    // Find cluster number
    int seq = 0;
    while ( i > 0 ) {
        i--;
        seq = seq * (maxChildren - 1) + idArray[i];
        }

        return seq;
    }

    // Calculates the number of nodes in this cluster.
    public int calculateNumClusterNodes( int clusterNumber, String[] args ) {
        int numClusters = Integer.parseInt( args[1] );
        return Integer.parseInt( args[numClusters + clusterNumber + 2] );
    }

    // Calculates the starting MPI rank of this cluster
    public int calculateClusterStartingRank( int clusterNumber, String[] args ) {
        int numClusters = Integer.parseInt( args[1] );
        int rank = 0;
        for (i=0; i < clusterNumber; i++) {
            rank += Integer.parseInt( args[numClusters + i + 2] );
        }
        return rank;
    }
```

Major changes to SentinelAgent.java:

// New version of initArgs() to handle cluster parameters

// New version of init() to handle child spawning with clusters