AgentTeamwork Design Sheet

Munehiro Fukuda[*]

Miriam Wallace[*]

[*] Computing and Software Systems, University of Washington, Bothell

AgentTeamwork Design Sheet

Table of Contents

Table of Figures

1. Commander Agent

This agent is responsible for launching all other AgentTeamwork system agents and all user ATeam programs.

Commander Agent uses functions provided by the `AgentUtil` class to manage utility operations shared across all AgentTeamwork agent classes (including but not exclusive to: counting the number of children, sending agent id and ip information to all agents in the tree, and receiving messages from children).

When invoked by the user, the Commander Agent is constructed on the local machine, following the execution model of the UWAgent class. Commander Agent's general constructor receives an array of arguments that indicate user options. After construction, the commander migrates to its host system and begins execution with the its `init` function.
The `init` function of the Commander Agent catches errors in construction and terminating the launch. If construction was successful, `init` spawns the necessary child agents (Sentinel, Bookkeeper, and Resource) using `spawnAll` as seen in *Figure 1*.

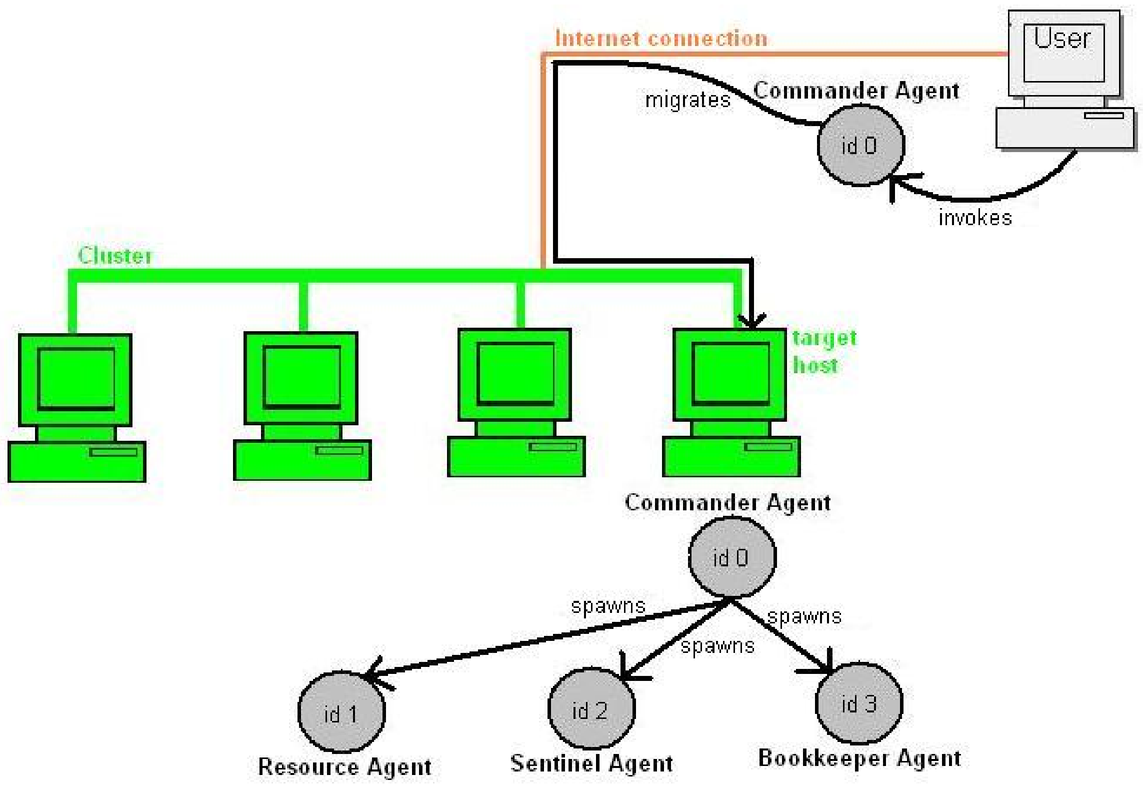


Figure 1. Remote user invoking Commander Agent for a host in a remote cluster

The `spawnAll` function begins with spawning a Resource Agent, immediately followed by a Sentinel Agent and Bookkeeper Agent when specified by the user (for more information about what these agents do once they are spawned see their sections on the following pages). When not specified by the user, the spawning of Sentinel and Bookkeeper Agents is delayed until the Commander Agent receives an itinerary from the the Resource Agent indicating what hosts are available to run the Sentinel and Bookkepper Agent tasks.

Once all the necessary child agents have been spawned, the Commander Agent uses the `mainMethod` function to ping children and wait for all agent and user processes to be completed before terminating.

2. User Program Wrapper and Sentinel Agent

2.1 Sentinel Agent

This agent is owns the User Program Wrapper containing and controlling the user program to be executed with the AgentTeamwork system.

When invoked by a Commander Agent, like other agents, the Sentinel Agent is constructed at the current location (with the Commander Agent) and then immediately transmitted to its target remote location (specified in the construction call) where initialization process begins.

The first Sentinel created by the Commander is the *root* Sentinel and spawns children as needed for current program execution as seen in *Figure 2*. Child Sentinels can be cluster gateways, public-desktop computing-nodes, or remote-cluster computing-nodes all of which have different spawning behavior. While all Sentinels maintain a list of nodes to assist in resumption, different Sentinel types can have different lists in order to allow for better performance across clusters. All Sentinels use an instance of the `AgentUtil` class to manage the user program's arguments and name.

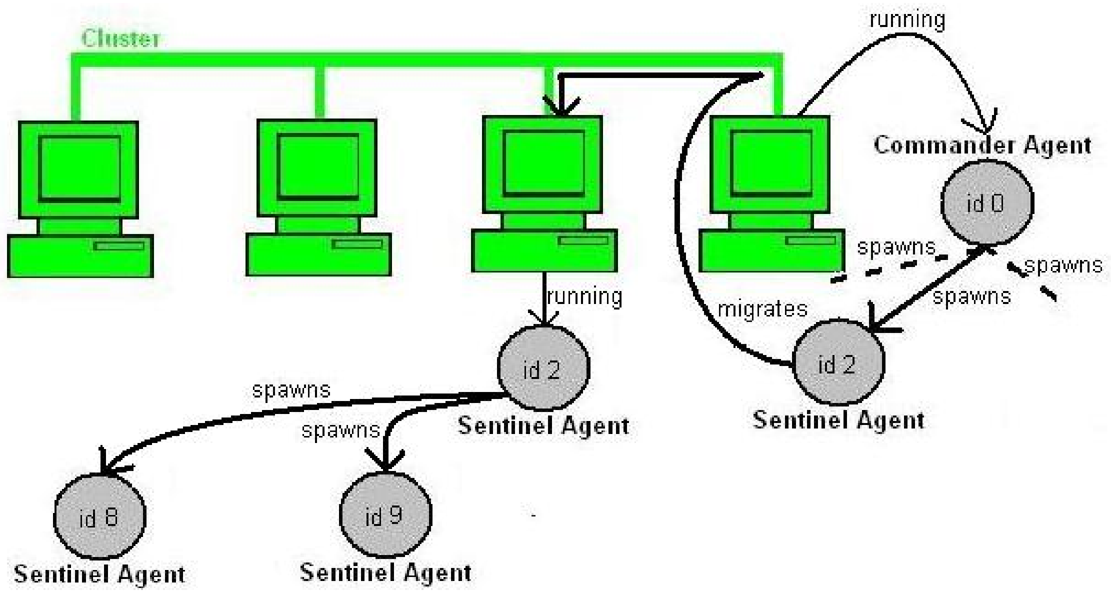
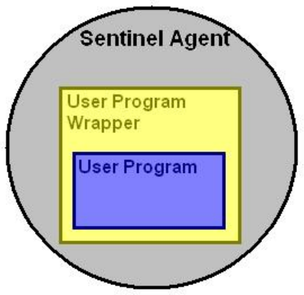


Figure 2. Root Sentinel Agent spawns children

When a Sentinel Agent is initialized it begins monitoring communications from other agents. Communications are monitored on an separate threads of execution from other agent work. New threads from the current agent process are established for tasks including: processing incoming communications; pinging other agents in case of crashes; and receiving output files.

Once these system threads have been spawned the Sentinel Agent calls the `funcMethod` function, which launches and monitors the given user program in the main thread of the executing Sentinel Agent by instantiating an instance of the User Program Wrapper and calling its `funcSch` function.

2.2 User Program Wrapper



User Program Wrapper captures and controls a user program execution within a Sentinel Agent in the AgentTeamwork execution platform as illustrated in *Figure 3*. Once the `funcSch` function is invoked by a Sentinel, the User Program Wrapper immediately saves a snapshot of its current state and then ensures that all processes are synchronized before invoking the user's program.

Arguments being passed to the user's program are passed to the User Program Wrapper at construction.

Figure 3. Sentinel posses User Program Wrapper which encapsulates User Program

3. Bookkeeper Agent

Bookkeeper Agents are responsible for storing and retrieving the snapshots of other agents using a hash table and the local disk allowing snapshotted agents to resume effectively after a crash.

The Bookkeeper constructor, much like other agents, takes the number of bookkeeper agents to be created from the Commander agent which instantiates it, and a list of host computers. Once constructed, the new Bookkeeper Agent migrates to its target host and completes initialization with the `init` function there.

The first Bookkeeper created by the Commander is the *root* Bookkeeper and spawns children as needed for current program execution as shown in *Figure 4*. All Bookkeepers use an instance of the `AgentUtil` class to manage the user program's arguments and name. New threads from the current bookkeeper agent process are created to handle receiving messages from other agents. Once initialization is complete, Bookkeepers continue to their `mainMethod`.

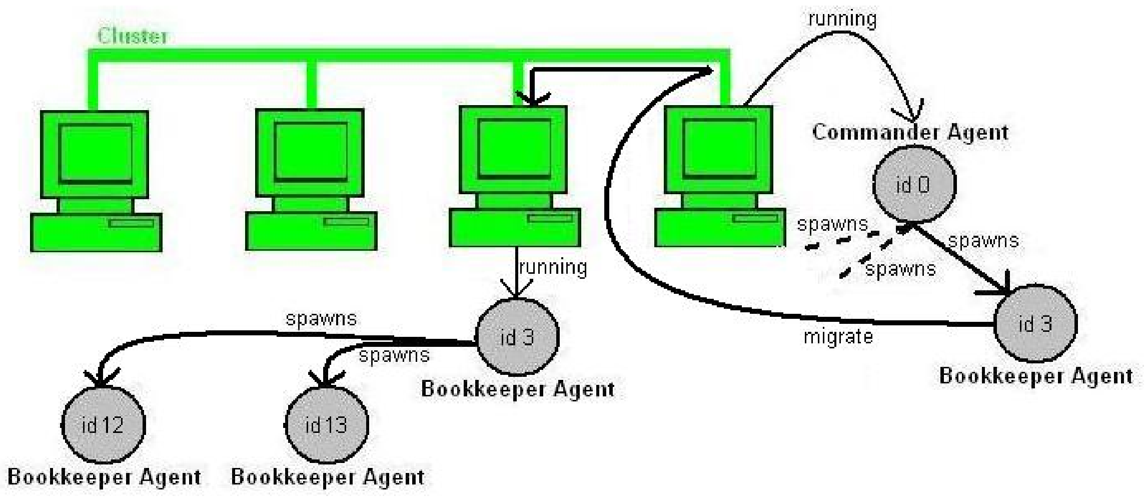


Figure 4. Root Bookkeeper spawns children to help store and retrieve agent execution snapshots

A Bookkeeper Agent's `mainMethod` tracks its left and right siblings (or neighbors) in order to send snapshots of itself to them in case recovery from a crash of the current agent is needed. The Bookkeeper then begins a loop where it continuously pings it's parents and siblings in order to detect a crash, while simultaneously waiting for incoming communication from other agents with its receive message thread.

When a Bookkeeper agent receives a communication from another agent, priority switches from pinging parents and children to processes the incoming message(s). Bookkeepers handle three special messages from other agents: messages to save a snapshot of a given Sentinel Agent; receipt of snapshots which are forwarded to the current Bookkeeper by a neighbor; and retrieving snapshots for agents when they are requested.

When a snapshot is received, the Bookkeeper Agent saves it to the local disk to assure its availability even if the Bookkeeper crashes and must resume. When a request to retrieve a snapshot is received, the Bookkeeper retrieves the snapshot from the hash table and sends it to the agent that requested the snapshot. If no snapshot is found, the Bookkeeper notifies the requesting agent of the failure.

Bookkeepers' most important function is the storing and retrieving of snapshots of other agents therefore, receiving and processing agents' communications is extremely important in any Bookkeeper Agent process.

4. Resource Agent and XML Database

4.1 Resource Agent

The Resource Agent spawned for a user program execution owns and maintains a database of resources available to the AgentTeamwork system and user programs. Resources include clusters and individual node computers which may be used in execution of a user's program. The Resource Agent is also responsible for allocation and deallocation of resources as needed by the system and uses specialized child Agents called Sensor Agents to report the status of remote nodes.

The Resource constructor takes an ftp address, user name and password for a remote database of resources to be integrated with any specific resources specified by the user at invocation of the AgentTeamwork system. Once constructed, the new Resource Agent migrates to its target host and completes initialization with the `init` function there.

The `init` function of a Resource Agent is responsible starting the local database of resources available to the user's program and updating resource data from the remote ftp XML Database files as shown in *Figure 5*. `Init` also creates a new thread to receive incoming communications from other agents.

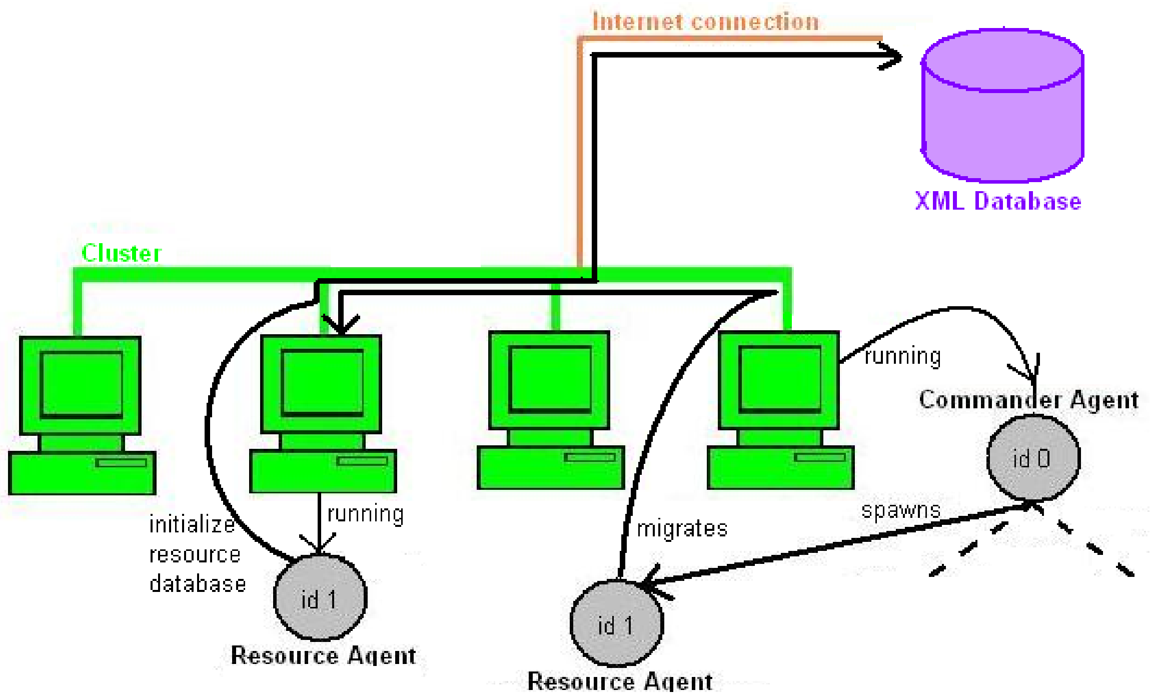


Figure 5. Resource Agent contacts the XML Database

Once initialization and updates are complete, the Resource Agent moves to its `mainMethod` where it begins probing available resources for the specified domain (if no domain has been specified it chooses a default domain of “UWB”). In order to make the best use possible of available resources, the Resource Agent spawns children called Sensor Agents which track usage across host clusters and computers and reports back to the Resource Agent on availability.

In the interest of managing accessible computing resources, the Resource Agent receive a few types of messages which other Agents do not, including notification of: need for additional resource allocation; call for deallocation of resources; previously accessible resources have been lost; and incoming status reports on remote nodes from Sensor Agents.

When the Resource Agent receives notice of a need for additional resources to be allocated, it searches known computing resources for a destination which meets the need indicated in the message. When appropriate resource is found the Resource Agent mark it allocated and responds to the requesting Agent with an itinerary indicating the location of exploitable resources.

When the Resource Agent receives a call for deallocation of resources it returns the resources to the pool of available resources held in its local database and marks those resources which are unresponsive with crashed status.

When the Resource Agent receives notice that previously accessible resources have been lost, it updates their status in its local database as failed, keeping the database’s list of accessible resources current for future allocations.

When the Resource Agent receives status reports from Sensor Agents the reports are processed and stored in the Resource Agents hash tables for reference.

4.2 XML Database

The XML Database contains information XML resource definitions for available nodes and clusters. A Resource Agent can use the information provided in the database to connect the AgentTeamwork platform with the resources provided therein, allowing agents to utilize these computing nodes for task completion.

Upon construction, the XML Database first checks its local file system (typically the ftp server it is located on) for information and then waits for a client connection with a resource agent. Once a connection is established the Resource Agent can update the database with information, as well as reference existing information on available resources.

In this way, the XML Database allows all AgentTeamwork processes to be kept updated with the best possible list of available resources from a central location.

5. Sensor Agent

The Sensor Agents are spawned by the Resource Agent to keep track of resources and to act as gateways to other networks. There are several types of Sensor Agents: Gateway; Client; and Server. The constructor collects and store invocation arguments, but the `init` function does the work of setting up the different Sensor types after the agent migrates to its first target location.

The `init` function establishes Gateway Sensors to link agents on remote computing nodes with other agents, and provides a conduit for two-way communication between remote nodes on a given cluster and other agents. Client and Server Sensors use TTCP to measure network throughput and communication performance data through the agent hierarchy back to the Resource Agent as illustrated in *Figure 6*. All Sensor agents also spawn a thread to listen for Agent communications. Once initialization of the different types of Sensotr Agents is complete, control passes to the `mainMethod`.

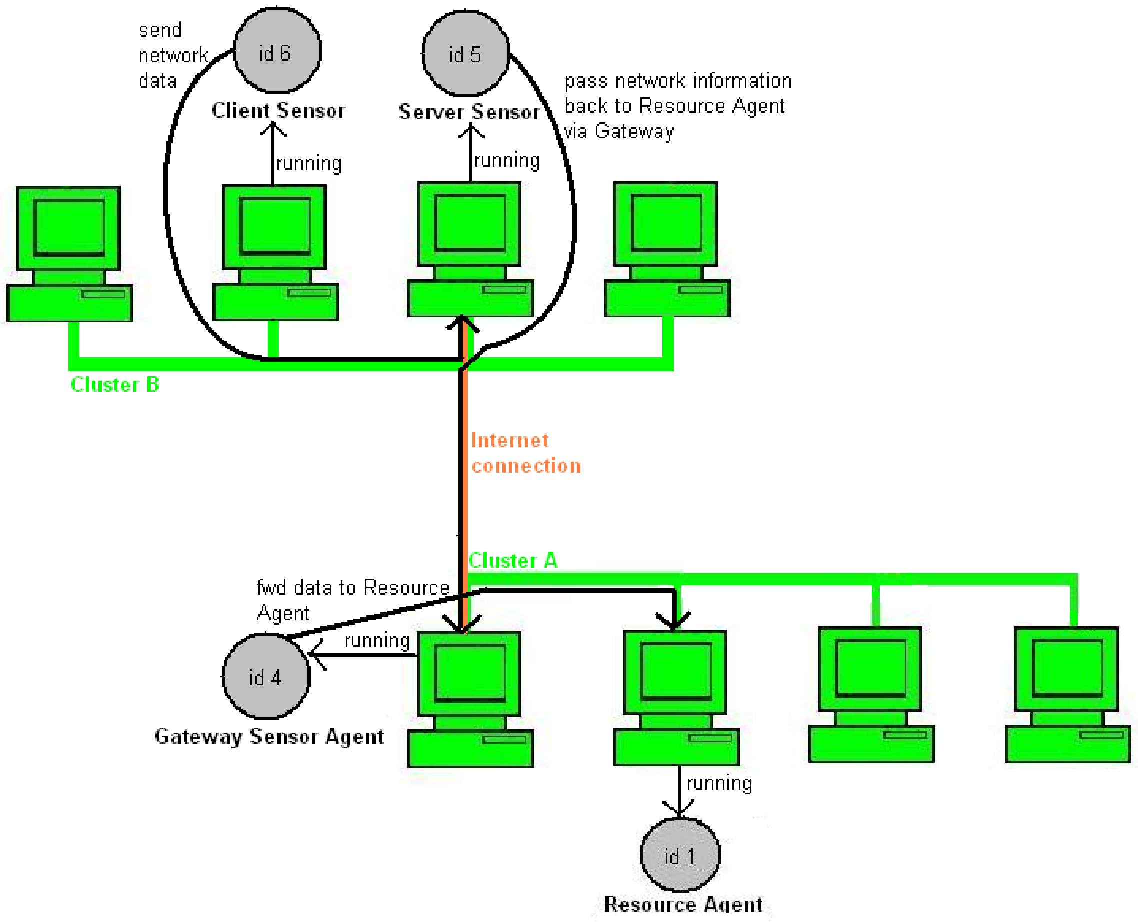


Figure 6. Sensor Agents probing another cluster

The `mainMethod` function Client and Server Sensors begin probing clusters for performance data using TTCP, and listening for a message signaling the agents to kill themselves.