CometCloud Developer Guide

How to implement your application using CometCloud

1) Download CometCloud source code (Refer to Deployment Guide). Create your own package in CometApps for your application and include lib/Automate/ jar files into your libraries. Refer to a sample application for writing codes in CometCloud-lite/comet_lite_src/application/src/tassl/automate/programmodel/masterworker/sample and implement your codes similar to sample application.

2) Starter, TaskTuple, Master, Worker classes are required. (Refer to `AppStarter.java`, `AppTaskTuple.java`, `AppMaster.java` and `AppWorker.java` in sample application)

3) `AppStarter.java`: It includes main to start your application. It calls `initComet` and `startApp` in `CommonApplicationStarter.` `initComet` is for setup the Comet infrastructure. Comet space is created and the nodes join the overlay. `startApp` starts your application. If the role of node is master, then master code runs, and if the role is worker then worker code runs. If you need to implement something between `initComet` and `startApp`, then implement it in `appSpecificStartup`.

4) `AppTaskTuple.java`: Task tuple is defined here. Implement at least three methods to set a task tuple, get a task tuple and get a query statement. (Refer to `setTaskStr`, `getTaskTuple` and `getQuery` in sample application.) When you set a task tuple, it is defined in the form of the following:

```
<AppTask>
      <TaskId> taskid </TaskId>
      <YourTag1> ValueOfYourTag1 </YourTag1>
      <YourTag2> ValueOfYourTag2 </YourTag2>
      …
      <MasterNetName> masterNetName </MasterNetName>
</AppTask>
```

   You can add your tags in a task tuple, and set a few of them as routing keys. Routing keys will be used for mapping the tuple to a node and make your search route to it. We suggest including task id and master net name or IP address as mandatory tags. Task id is for identifying each task and master net name is used for sending results back to master. Implement at least `setTaskStr`, `getTaskTuple` and `getQuery`. For a search query, you can specify the value of a tag as well as use range query by '*'.

5) `AppMaster.java`: Build your master implementing `MasterFramework` class. You should overwrite interfaces by `MasterFramework`. After a node joins the overlay, basic Comet information such as Comet space, overlay, master id and IP address is passed to the master class. Set this environment in your master class using `setCometEnv`. Fill in `startMaster` method to run your master. For example, if you implement it as a thread, then start the thread here. `setResult` method is called whenever the master receives a result from workers. Implement your code for managing each result.

Master code should contain task generation. We suggest making task generator as a thread if the master has to generate a lot of tasks because the master can receive results during generating tasks. To generate a task, follow the steps:

a. Make a task string using a method for setting a task tuple in `AppTaskTuple`.

b. Create an XmlTuple.
```
XmlTuple task = new XmlTupleService();
task.createXMLtuple(task_string);
```

c. After generating data for a task, serialize it and attach it to the task tuple.
```
Object your_data;
//generate your data
byte data[] =
programming5.io.Serializer.serializeBytes(your_data);
task.setData(data);
```

d. Use '`out`' method for sending the task to the Comet space.
```
cometspace.out(MWConstants.spacename, task, taskid,
data, peerIP, overlays);
```
where cometspace, peerIP, overlays were set by `setCometEnv` method.

A few of tasks can be lost by some reasons such as network congestion and failure. `TaskMonitoring` class is provided for checking task status and reinserting lost tasks. You need to implement `public int[] getTaskStatus()`, `public int getTaskStatus(int taskid)`, `public int getNumOfTasks()` and `public void reinsertTask(int taskid)`. Taskstatus is defined by integer array in the sample application, but you can use a different type. Override `public int getTaskStatus(int taskid)` for returning the status of a task. Initially taskstatus of each task is set to 0 and becomes 1 when the master gets its result. To start task monitoring, make an instance of `TaskMonitoring` class, and start the thread. To terminate it, call `quit()` method.

6) `AppWorker.java`: Build your worker implementing `WorkerFramework` class. You should override `setCometEnv`, `startWorker`, `computeTask` and `sendResultToMaster`. After a node joins the overlay, basic Comet information such as Comet space, overlay, master id and IP address is passed to the worker class. Set this environment in your worker class using `setCometEnv`. Fill in `startWorker` method to run your worker. For example, if you implement it as a thread, then start the thread here. Implement what a worker should do after it gets a task in `computeTask`. `sendResultToMaster` method is called whenever the worker sends the result back to the master after it finishes a task. Workers repeat to get a task from the Comet space, consume it and send the result back to the master. Steps are the following for consuming a task:

a. Get query statements using a method defined in AppTaskTuple.
```
XmlTuple queryTuple = new XmlTupleService();
queryTuple = AppTaskTuple.getQuery();
```

b. Use '`in`' method for getting a task from the Comet space.

```
List taskList = cometspace.in(MWConstants.spacename,
queryTuple, your_query_option, Long.MAX_VALUE, peerIP,
overlays);
```

Where cometspace, peerIP, overlays were set by `setCometEnv` method. As a query option, you can use `TSConstant.GET_ANY` for getting one task matching your query or `TSConstant.GET_ALL` for getting all tasks matching your query.

c. Deserialize data.

```
Object obj =
 programming5.io.Serializer.deserialize(databyte);
```

d. Add your code for work with data.

e. Send the result back to the master.

```
public void sendResultToMaster(int taskid, Object data,
String message, String masterName)
```

Required properties files

1) chord.properties
   a. ID_BITS : Number of bits for chord ID. A larger value is required for more routing keys.

   b. LOCAL_URI: Address of the node on the local machine in the form //host:port. Each different node run on the same machine must have a different port. If you don't set this property, then it will be set automatically inside of Comet. If you run different node on the same machine, port number starts from one that you set as CometPort in comet.properties.

   c. LOCAL_CLUSTER: Used for two-level chord

   d. REMOTE_BOOTSTRAP_LIST: Used for two-level chord

   e. STABILIZE_PERIOD and FIX_FINGERS_PERIOD: Periodicity of self-healing tasks. Disabled when commented out.

2) squid.properties
   a. SPACE_DIMENSIONS: Number of index dimensions. Set the number of routing keys that you use.

   b. BIT_LENGTH: Number of bits used to encode the values of each dimension. (Note: SPACE_DIMENSIONS x BIT_LENGTH should equal to chord.ID_BITS property)

   c. KEY_TYPE: One of NUMERIC and ALPHABETIC. Set the type of each routing key. This property applies for each dimension i (prefixed by Di). i begins from 0.
   example) squid.D0.KEY_TYPE=NUMERIC
   squid.D1.KEY_TYPE=ALPHABETIC

3) comet.properties
    a. MasterClass, WorkerClass, TaskClass: Specify master, worker, and task tuple class of your application

    b. RoutingKeys: One or more of XML tags which you defined in the task tuple. More keys you use, larger key space and Chord ID_BITS are required. This makes key space large and increase the overhead. Set this property to the appropriate tags which you want to use for search queries.

    c. TaskMonitoringPeriod: Master checks space every TaskMonitoringPeriod to regenerate missing tasks. Default is 10000 ms.

    d. OUTCONTROLLER: Set true or uncomment it if you want CometCloud to buffer tasks generated by the master instead of sending them out to the space immediately. This is for reducing the used amount of memory of the space. The master checks the space every OUTCONTROLLER_CHECKPERIOD and if the number of tasks in the space is lower than OUTCONTROLLER_LOWER, then the master sends OUTCONTROLLER_OUTTASK tasks out to the space.

    e. ReplicationEnable: Set this property to false to disable replication. Default is true.

    f. IsolatedProxy: If you run isolated workers which do not share the Comet space and just provide computing capabilities, then set the proxy.

    g. Scheduler, SchedulerClass: Set scheduler with ip address (or hostname):port number and scheduler class for enabling autonomic cloudbursts and cloud bridging over multiple clouds.

4) nodeFile
    a. All nodes joining the overlay should be described here by IP address (or hostname):the number of nodes.
    b. Example)
    111.111.111.111:3

5) portFile
    a. Describe port numbers here. At least the maximum number of ports of a node described in nodeFile should be defined here. If the maximum number of nodes of a node in nodeFiles is n, then n ports should be specified here.
    b. Example)
    5555
    5556
    5557

6) exceptionFile
    a. Describe the role of each node as follows. If nothing is described, then the node will be a worker.
    b. Example) 111.111.111.111:5555 is a master, 111.111.111.111:5556 and 111.111.111.111:5557 are workers.
    111.111.111.111:5555
    comet.NodeType=MASTER

How to run your application using CometCloud

1) Set chord.properties, squid.properties, comet.properties in the master node.

2) Set nodeFile, portFile, exceptionFile in the master node

3) Run overlayControlServer on all nodes described in nodeFile
   java -cp $CLASSPATH tassl.automate.overlay.OverlayControlServer 4444

4) Run your application only on the master. All other nodes described in nodeFile will automatically start.
   java -cp $CLASSPATH tassl.automate.application.your.application.starter -nodeFile nodeFile -portFile portFile -exceptionFile exceptionFile -propertyFile chord.properties -propertyFile squid.properties -propertyFile comet.properties

How to run your application with isolated workers

1) If you run clouds such as Amazon EC2 and rent only computing capabilities from them without sharing the Comet space, then run isolated workers.

2) You should not describe isolated workers in nodeFile.

3) Set `IsolatedProxy` in comet.properties where isolated workers run.

4) Create RequestHandlerList specifying IP addresses of one or more request handlers where a proxy runs.
   a. Example) RequestHandlerList
      111.111.111.111

5) Specify your request handler(s) in exceptionFile
   a. Example) modified exceptionFile.
      111.111.111.111:5555
      comet.NodeType=MASTER
      111.111.111.111:5556
      comet.NodeType=REQUEST_HANDLER
   b. Note that at least master and request handler(s) should join the overlay even in running isolated workers.

6) Run overlayControlServer on all nodes described in nodeFile
   java -cp $CLASSPATH tassl.automate.overlay.OverlayControlServer 4444

7) Run proxy
   java -cp $CLASSPATH tassl.automate.application.node.isolate.RequestHandlerProxy

8) Run your application only on the master. All other nodes described in nodeFile will automatically start.
   java -cp $CLASSPATH tassl.automate.application.your.application.starter -nodeFile nodeFile -portFile portFile -exceptionFile exceptionFile -propertyFile chord.properties -propertyFile squid.properties -propertyFile comet.properties

9) Run isolated worker

```
java -cp $CLASSPATH tassl.automate.application.node.isolate.CloudBurstStarter -propertyFile comet.properties
```