Storm Adaptive Schedulers – User Manual

This document contains the instructions for compiling, deploying and using the general-purpose adaptive schedulers for Storm (http://storm-project.net/), developed by MIDLAB research group (http://www.dis.uniroma1.it/~midlab/). For detailed info about the software, please refer to

> Leonardo Aniello, Roberto Baldoni, Leonardo Querzoni,
> "Adaptive Online Scheduling in Storm",
> 7th ACM International Conference on Distributed Event-Based Systems (DEBS 2013),
> available online at http://www.dis.uniroma1.it/~midlab/articoli/ABQ13storm.pdf.

Requirements

Current implementation only works in UNIX-like environments.

Where to download

The zip file containing the source code is available online at
http://www.dis.uniroma1.it/~midlab/software/storm-adaptive-schedulers.zip.

How to compile

The `src` directory in the zip contains all the sources required for the compilation. The external libraries to include are

- Storm 0.8.1 libraries
  (https://dl.dropbox.com/u/133901206/storm-0.8.1.zip)
  - `storm-0.8.1.jar`
  - `lib/*.jar`
- Commons DBCP 1.4
  (http://archive.apache.org/dist/commons/dbcp/binaries/commons-dbcp-1.4-bin.zip)
  - `commons-dbcp-1.4.jar`
- Commons Pool 1.5.6
  (http://archive.apache.org/dist/commons/pool/binaries/commons-pool-1.5.6-bin.zip)
  - `commons-pool-1.5.6.jar`

It is recommended to compile using Java 1.6.x.

How to deploy

Generally, a Storm deployment includes a *master node* (where nimbus service runs) and several *slave nodes* (where supervisor services run). In each node (either master or slave), these jars have to be copied into the `lib` directory of Storm installation

- the jar resulting from the compilation of Storm adaptive schedulers
- `commons-dbcp-1.4.jar`

- commons-pool-1.5.6.jar

The online scheduler needs a MySQL database to store and retrieve online statistics about inter-executor traffic and CPU load. The required MySQL version is 5.5.x. The SQL script storm-scheduler.sql included into the zip is for creating all the required DB tables. A MySQL Java connector 5.1.x is required at runtime to enable the interaction with the DB. It can be downloaded from http://downloads.mysql.com/archives.php?p=mysql-connector-java-5.1&o=other and has to be copied into the lib directory of Storm installation, for each node (either master or slave).

In order to configure the master node, two files have to be modified: storm.yaml (into conf folder of Storm installation) and db.ini (to be put into the root directory of Storm installation). Next two tables describe how to configure such files.

storm.yaml
In this configuration file, the format to use for the single parameter-value pair is
parameter: "value"
All these parameters are mandatory

Parameter	Description	Allowed values
storm.scheduler	Specifies the fully qualified name of the class to use as scheduler	For the offline scheduler: midlab.storm.scheduler.OfflineScheduler
		For the online scheduler: midlab.storm.scheduler.OnlineScheduler
traffic.improvement	Specifies the minimum percent inter-node traffic improvement required to apply a new schedule[1]	An integer between 1 and 100 (for online scheduler only)
reschedule.timeout	Specifies the minimum amount of time (in seconds) that has to pass before a new schedule can be applied	A positive integer (for online scheduler only)

db.ini
In this configuration file, the format to use for the single parameter-value pair is
parameter=value
All these parameters are mandatory

Parameter	Description	Allowed values
jdbc.driver	Specifies the fully qualified name of the class to use as DB connector	If MySQL 5.5.x is used, then the correct value is com.mysql.jdbc.Driver
data.connection.uri	Specifies the DB connection URI	If MySQL 5.5.x is used, then the correct value is (it's a single value to be written within a single line) jdbc:mysql://<DB_IP_ADDRESS>/storm-scheduler?user=<USER>&password=<PWD> where • <DB_IP_ADDRESS> is the IP address of the host where MySQL DBMS is running

[1] Let T be current inter-node traffic, and T' the estimate of inter-node traffic produced by a new schedule S.
If (T – T') / T ≥ traffic.improvement / 100, then schedule S is applied.

		• <USER> is the username to use for the connection • <PWD> is the password of the specified username
validation.query	Specifies the query to use for checking whether a DB connection is still alive	SELECT

In order to configure a slave node, only db.ini (to be put into the root directory of Storm installation) has to be configured. In addition to the three parameters previously described in the table for the db.ini for the master node (jdbc.driver, data.connection.uri, validation.query), others have to be configured and are described in the next table.

db.ini
In this configuration file, the format to use for the single parameter-value pair is
parameter=value
All these parameters are mandatory

Parameter	Description	Allowed values
node-name	Specifies an identification name (unique ID) for the slave node	Any string, with the constraint that any two distinct slave nodes are assigned distinct names
capacity	Specifies the maximum percentage of CPU usage allowed for this slave node[2]	An integer between 1 and 100
time.window.slot.count	Specifies the number of slots for a single time window[3]	A positive integer
time.window.slot.length	Specifies the length (in seconds) of a single slot of a time window	A positive integer

How to use

Some simple lines of code have to be added to spout/bolt source in order to enable the monitoring of inter-executor traffic and CPU load. In the specific each task has to notify its thread ID to a singleton object in charge of monitoring all the tasks running in a specific Java process (Storm worker).

For each spout/bolt, an instance field has to be added

```
private TaskMonitor taskMonitor;
```

Some initialization operations have to performed in the open() method of each spout

```
public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    ....
    // register this spout instance (task) to the java process monitor
    WorkerMonitor.getInstance().setContextInfo(context);
    // create the object required to notify relevant events (also notify thread ID)
    taskMonitor = new TaskMonitor(context.getThisTaskId());
    ...
}
```

[2] The total capacity of a node is measured in MHz by looking at CPU characteristics; the CPU demand (in MHz) for each executor is tracked in order to estimate the expected CPU load for each slave node when a new schedule S is applied.
[3] The measurements of inter-executor traffic and CPU load are carried out on a sliding time window: each time window is divided into a fixed number of equal-length slots, and the window slides by a single slot per time.

The same has to be done in the `prepare()` method of each bolt

```
public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {
    ...
    // register this spout instance (task) to the java process monitor
    WorkerMonitor.getInstance().setContextInfo(context);
    // create the object for notifying relevant events (received tuple, which in turn
notifies thread ID)
    taskMonitor = new TaskMonitor(context.getThisTaskId());
    ...
}
```

The notification of thread ID for the spouts is done in the `nextTuple()` method

```
public void nextTuple() {
    taskMonitor.checkThreadId();
    ...
}
```

Finally, the notifications of received tuples for the bolts is done in the `execute()` method

```
public void execute(Tuple input) {
    taskMonitor.notifyTupleReceived(input);
    ...
}
```

Other parameters can be configured for each topology. Their names vary a little with respect to what is written in the paper referenced at the beginning of this document. Available parameters are described in next table. These parameters have to be specified in the source code of the topology: they have to be put into a `java.util.Map` instance that has to be passed to `submitTopology()` method.

Topology parameters

Let `conf` be a `java.util.Map` instance, the code to use for the single parameter-value pair is `conf.put(parameter, value)`

For a generic topology t_i, these variables are defined

- E_i: number of executors
- W_i: number of workers
- C_i: number of components (spouts/bolts)

The number of slave nodes in the cluster is N.

Parameter	Description	Allowed values
`alfa`	Controls the balancing of the number of executors assigned per slot. In particular, `alfa` affects the maximum number M of executors that can be placed in a single slot. The minimum value of M for a topology t_i is $\lceil E_i / W_i \rceil$, which means that each slot roughly contains the same number of executors. The maximum number of M corresponds to the assignment where all the slots contain one executor, except for one slot that contains all the other executors, so its value is $E_i - W_i + 1$. $M(alfa) = \lceil E_i / W_i \rceil + alfa(E_i - W_i + 1 - \lceil E_i / W_i \rceil)$	A float in the range [0,1]
`beta`	Controls the number of nodes to use. It works similarly to `alfa`. If `beta = 0`, then $min = \lceil W_i / N \rceil$ nodes are used. If `beta = 1`, then $max = min(W_i, N)$ nodes are used. In general, `min + beta * (max - min)` nodes are used.	A float in the range [0,1]
`gamma` (only for online scheduler)	Controls the maximum load acceptable by a slot, it is computed as `gamma * (totalLoad / Wi)`, where `totalLoad` is the sum of loads of all the executors of topology t_i.	A float greater than 1

`delta`	Controls the maximum number of slots per node. The minimum value is $\texttt{min = ceil}(W_i / N)$. The maximum value if $\texttt{max} = W_i - N + 1$. The actual value is computed as `min + delta * (max - min)`.	A float in the range [0,1]
`epsilon` (only for offline scheduler)	Controls when to start considering empty slots. During the assignment of executors for the i^{th} component, the scheduler is forced to use empty slots if $i > \texttt{floor}(\texttt{delta} * C_i)$. For example, if traffic is likely to be more intense among upstream components, then delta should be set large enough such that empty slots get used when upstream components are already assigned.	A float in the range [0,1]