Documentation for master's thesis

1 Main branch and usage

The final branch containing the two most useful configurations is called *distributed-final*. To build this branch, operators must be install with branch *custom-tweetformat-windowstate*. There are two main settings:

- Global batching:
  Wait until N retweets have been received or until M milliseconds have passed. Then reconstruct all cascades that were part of this batch.
- Batching per cascade:
  Wait until N retweets have been received. Then perform reconstruction for this cascade.

1.1 Commands

Absolute paths should be used to access files on hard disk.

1.1.1 Basic parameters

These parameters are needed for every configuration:

- -f <JSON file containing the dataset>
- -o <folder to JSON file>
- -nodeCount <number of nodes for reconstruction>
- -sgmDirectory <folder containing social graph data>
- -sgType [friends/followers]
- -sgLoadMode archives
- -maxHeapNode <max heap space per worker process>
  choose something like $total\ main\ memory - 2G$, e.g. 30G for DBIS cluster
- -nodeDirectory <folder containing social graph partitioning>
- -algo [incHashNParentFinder/incUserIterationParentFinder]
- -deployName <name of topology>

1.1.2 Batch processing parameters

Default is cascade batching with batch size 10.

- -globalBatching <batch size> global batch processing
- -globalBatchingTime <milliseconds> global batching after a given number of milliseconds. Can be combined with other -globalBatching option. Reconstruct, if any condition is satisfied.
- -cascadeBatching <batch size> batch processing per cascade

1.1.3 Optional parameters

- -remoteRoutingThreads <number of threads for RemoteRoutingBolt> beware that the machine running RemoteRoutingBolt must run three other executors: Spout, WindowBolt, WindowIncrementalBolt
- -reconstThreads <number of threads for ReconstructionBolts>
- -writeResults [true/false] turn writing of influence edges on/off
- -padUserlists should always be given when UserIteration is used for reconstruction
- sgLoadLogging [true/false] turn warning during loading of social graph on or off

Attention: Multithreading cannot be used together with any kind of global batching.

1.2 Deployment on DBIS cluster

The application runs on sydney and dbisma02-dbisma10. Dbisma01 is left free, because it runs monitoring tools of the cluster. The scripts being used are all located in */home/lutzb/bin*. Copy these files somewhere and add this folder to your path. Sydney runs the Nimbus and the Storm UI, since it can be accessed from outside the network of the university. The UI can be accessed via: *http://isydney.informatik.uni-freiburg.de:13337/index.html* Execute the following steps, to initialize the Storm cluster:

- Start zookeeper: execute *startZookeeper* on dbisma10
- Start Nimbus and Storm UI: execute *startNimbus* on sydney

- Start Supervisors: execute *startSupervisor* on dbisma02-dbisma10

Now, the cluster is ready to run. I suggest creating the following aliases on dbisma02-dbisma10 to simplify debugging:

```
# look for exceptions
alias ex='grepAllLogs Exception|grep -v Netty'
# check whether social graph is already loaded
alias loaded='grepAllLogs loaded'
```

Otherwise with *grepAllLogs* all logfiles can be grepped for a given string. The options -n and -H are automatically given, so that the corresponding file and line number will be displayed. Execute *loginNodes &* on sydney to login with cssh on dbisma02-dbisma10.

1.2.1 Folder structure

Every node must have exactly the same folder structure for cascade input files, social graph data and social graph partitioning. Structure on the nodes:

```
/vol2/storm-distrib/
    sgfriends
    sgfollowers #sg data only on dbisma02-dbisma09
    tempdir-<dataset>
        retweeters.csv
        nodedistribution_8_edge_friends
            node0.csv
            ...
            node7.csv
        nodedistribution_8_edge_followers
        nodedistribution_8_truerandom
        nodedistribution_8_metisag
        nodedistribution_8_metissg
    <dataset>-proj #JSON files contained only on dbisma10
/home/lutzb/storm-local-dir/
    reconstBolt<nodeId>_<threadId>.log # dbisma02-dbisma09
    routing<threadId>.log # dbisma10
/home/lutzb/bin
    clearLogs
    clearResults
    startSupervisor
    grepAllLogs
```

Hint: bigger5 corresponds to dataset *bigger4* in thesis and bigger1000 corresponds to dataset *1k-30k* in thesis. On eath node, copy all files in */home/-lutzb/bin* somewhere and add this folder to your PATH.

1.2.2 Collecting logfiles and extracting results

When there are no more changes on the UI in terms of tuples processed for the StorageBolts or if you are sure, that the dataset was processed completely, execute the following command on sydney:

```
aggregateMixCascadesSeq <outputFolder>
```

This will download the logfiles of all nodes into *outputFolder*, iterate over them and aggregate the results. The results will be printed to stdout.

1.3 Deployment on Amazon EC2

Attention: nodes have Java 1.7 installed! Use storm-deploy-alternative to deploy Storm cluster on EC2. Everything must be correctly configured before, because configuration cannot be changed afterwards. Use the following configuration file, to create a cluster consisting of nine *r4.2xlarge* nodes. StormScheduler must be available at some url, that can be accessed from everywhere.

```
experiment:
    - storm-version "1.0.1"     # Version of Storm
    - zk-version "3.4.6"     # Version of Zookeeper
    - image "us-east-1/ami-60b6c60a"    # Amazon Linux / CentOS
    - image-username "ec2-user"
    - packagemanager "yum"
    - region "us-east-1"     # Region
    - mount-local-storage "false" # set to false if EBS is used
    - mount-ebs-storage-size "50" # delete for instance with HDD
    - private-key-path "/path/to/snsa_cluster"
    - public-key-path "/path/to/snsa_cluster.pub"
    - scheduler-webdownload-path "https://www.dropbox.com/s/8
      eixdix9kqurmtn/StormScheduler-0.0.1-SNAPSHOT.jar?dl=1"
    - r4.2xlarge {ZK, WORKER, MASTER, UI}    # master node
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - r4.2xlarge {WORKER}
    - remote-exec-preconfig {cd ~, echo hey > hey.txt}
    - remote-exec-postconfig {}
```

There are two possibilities for local storage:

- EBS storage: Set *mount-local-storage* to „false“, set *mount-ebs-storage-size* option to desired amount
- local HDD: Set *mount-local-storage* to „true“ and delete *mount-ebs-storage-size* option

Execute main method in StormDeployAlternative, with parameters:

```
deploy experiment
```

This takes some time. Application will print IPv4 addresses of nodes to stdout. Copy these addresses and paste them into some file, e.g. *amazonNodes*. Copy folder *amazon_scripts* to folder */mnt* on all nodes. Then, execute following commands on all nodes.

```
export PATH=$PATH:/mnt/amazon_scripts
amazonNodesLoadS3
```

Friends and distributions will be downloaded from S3. (Followers are commented out) Then, execute following command on master node.

```
amazonNodesLoadS3Leader
```

This will download and untar the cascade files from S3. Next: slave nodes must tell their IP address to master node:

```
tellLeaderIP <IP of master>
```

Finally, copy snsa-storm-twitter-analysis.jar to master node. Deploy topology with

```
reconstructCascadeFriendsAmazon <dataset> <distribution> <jar file
    > [OTHER OPTIONS]
```

After topology is considered finished, execute following command on master node to collect and aggregate metric files:

```
aggregateMixCasadesSeq <outputfolder>
```

1.4 Examples for common settings

Incremental reconstruction with batches per cascade, batch size 10, User Iteration on cluster of DBIS chair for dataset *bigger4*, using 3 threads for Routing- and ReconstBolts.
Execute on sydney:

```
storm jar snsa-storm-twitteranalysis-0.0.4-SNAPSHOT-jar-with-
    dependencies.jar de.unifreiburg.informatik.websci.topology.
    CentralCascWindowReconstTopology
    -maxHeapNode 30G
    -sgLoadMode archives
    -algo incUserIterationParentFinder
    -sgInfo friends -f /vol2/storm-distrib/bigger5-proj/bigger5.
       json
    -sgLoadLogging False
    -nodeCount 8
    -nodeDirectory /vol2/storm-distrib/tempdir-bigger5/
       nodedistribution_8_truerandom
    -deployName dist-reconstruct-bigger5-friends-random
    -sgmDirectory /vol2/storm-distrib/sgfriends/
    -reconstThrads 3
    -remoteRoutingThreads
    -cascadeBatching 10
    -padUserlists
    -o /vol2/storm-distrib/bigger5-proj
```

Before running experiments, the logfiles and previous results should be cleared. This can be done with the commands *clearLogs* and *clearResults <absolute path to cascade >*.
You can also use the following command, which will cleanup logs and previous results and then deploy a topology with standard parameters:

```
reconstructCascadeFriendsCustom <dataset> <distribution> <jarfile>
     [OTHER OPTIONS]
```

This will run the dataset with given distribution and jar file with default settings. Other options must still be given explicitly. For example:

```
reconstructCascadeFriendsCustom bigger5 random snsa-storm-
    twitteranalysis-0.0.4-SNAPSHOT-jar-with-dependencies.jar -
    reconstThrads 3 -remoteRoutingThreads -cascadeBatching 10
```

2 Other branches

In all previous configurations, PreParseSpout and SimpleWindowBolt were implemented so that configurations can be better compared. Besides, metrics printing was updated to version of final branch. Use only the basic parameters to run the following configurations.

2.1 State of the system before work started

The system at state of June 2016 is contained in branch *distributed-improved.* Operators must be in master branch. Prefix Iteration is used with blocking reconstruction and old datasets.

2.2 Blocking distributed reconstruction, User Iteration with old datastructure

The state of the system just with User Iteration algorithm enabled but old datastructures and no incremental reconstruction is contained in branch *distributed-followers-incremental.* Operators must be in master branch.

2.3 Blocking distributed reconstruction with optimized datastructure

The system with User Iteration and optimized datastructures is contained in branch *distributed-all-nobatch-windowstate.* Operators must be in branch *custom-tweetformat-windowstate.*

3 Code documentation

All branches are somehow of the same structure. In this section, the code of the branch *distributed-final* will be described.
The main class where the topology is built is called *CentralCascWindowReconstTopology*.
The following figure shows the topology of the application and the names of the classes marked in blue that represent each operator.

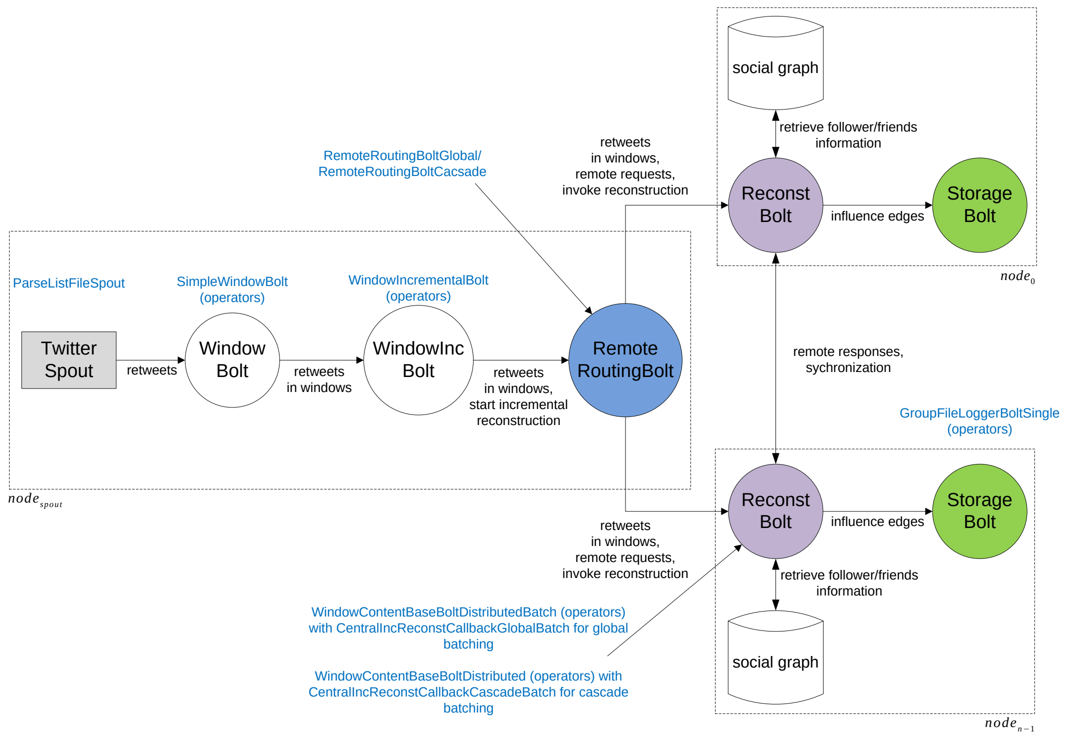
social graph
retrieve follower/friends information
RemoteRoutingBoltGlobal/ RemoteRoutingBoltCacsade
retweets in windows, remote requests, invoke reconstruction
Reconst Bolt
influence edges
Storage Bolt
$node_0$
ParseListFileSpout
SimpleWindowBolt (operators)
WindowIncrementalBolt (operators)
Twitter Spout
retweets
Window Bolt
retweets in windows
WindowInc Bolt
retweets in windows, start incremental reconstruction
Remote RoutingBolt
remote responses, sychronization
GroupFileLoggerBoltSingle (operators)
$node_{spout}$
Reconst Bolt
influence edges
Storage Bolt
retweets in windows, remote requests, invoke reconstruction
retrieve follower/friends information
WindowContentBaseBoltDistributedBatch (operators) with CentralIncReconstCallbackGlobalBatch for global batching
WindowContentBaseBoltDistributed (operators) with CentralIncReconstCallbackCascadeBatch for cascade batching
social graph
$node_{n-1}$

Figure 1: Topology with classes marked in blue for each operator

3.1 RemoteRoutingBolt

RemoteRoutingBolt is implemented in two versions, depending on how batch processing is used with incremental reconstruction. Depending on this setting, either *RemoteRoutingBoltGlobal* or *RemoteRoutingBoltCascade* will be instantiated. The *RemoteRoutingBolt* communicates with the reconstruction nodes via two types of channels. Lets assume, that we have N reconstruction nodes. The channels used for transferring retweets are called *0 to N- 1*, one for each reconstruction node. The channels used for issuing remote requests are called *Request0 to RequestN-1*.

RemoteRoutingBolt contains four abstract methods. Main functionality is in the *execute()* method.

```java
public abstract class RemoteRoutingBolt {
  public void execute(Tuple input) {
  ...
  }
  // called when a cascade is finished
  protected abstract void cascadeFinished(long windowId,
      Tuple input);

  // called at the beginning of a cascade, so that remote
      requests buffers can be initialized
  protected abstract void initRemoteRequestBuffer(long
      windowId, long rootUserId);

  // reconstruct should be performed, flush remote requests
  protected abstract void flushRemoteRequests(Tuple input);

  // add remote requests to buffer
  protected abstract void addRemoteRequests(long windowId,
      int ws, int rootNodeStream, long userId,
      Tuple input);
}
```

3.2 ReconstructionBolts

The ReconstructionBolts are inherited from *CentralIncReconstCallback*. The main function is *computeContents()*

```
public abstract class CentralIncReconstCallback {
  public List<Values> computeContents(List<Object> content,
      int state, long windowId) {
  ...
  }
  protected abstract List<Values> handleRemoteRequest(List<
      Object> content, long windowId);

  protected abstract List<Values> handleRemoteResponse(List<
      Object> content, long windowId);

  protected abstract List<Values> handleRemoteFlushResponse(
      List<Object> content, long windowId);

  protected abstract List<Values> handleRemoteFlush(List<
      Object> content, long windowId);;
}
```

The methods' purposes are self-explaining. They are called whenever a message with a certain has been received. There are two direct subclasses of *CentralIncReconstCallback*:

```
public abstract class CentralIncReconstCallbackCascadeBatch {
  protected abstract void insertRemoteEdges(long windowId,
      ArrayList<Long> parents, ArrayList<Long> childs);

  protected abstract List<Values> answerRemoteRequest(List<
      Object> content, long windowId);
}
```

Depending on whether User Iteration or Prefix Iteration is used for reconstruction, the corresponding subclass *CentralIncReconstCallbackCascadeBatchFriends* or *CentralIncReconstCallbackCascadeBatchFollowers* will be instantiated.

The other subclass of *CentralIncReconstCallback* is called *CentralIncReconstCallbackGlobalBatch*:

```
public abstract class CentralIncReconstCallbackGlobalBatch {
  protected abstract List<Values> answerRemoteRequest(
      ArrayList<Long> userIds, ArrayList<Integer> rootNodes,
      ArrayList<Long> windowIds);

    protected abstract void insertRemoteEdges(ArrayList<Long>
        parents, ArrayList<Long> childs, ArrayList<Long>
       windowIds);
}
```

For this class, the methods *handleRemoteFlushResponse()* and *handleRemoteFlush* are empty, because in global batching no explicit sychronization is needed.
Again, depending on the reconstruction algorithm, either *CentralIncReconstCallbackGlobalBatchFriends* or *CentralIncReconstCallbackGlobalBatchFollowers* will be instantiated.