Architecture of Enterprise Applications 27 Storm

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

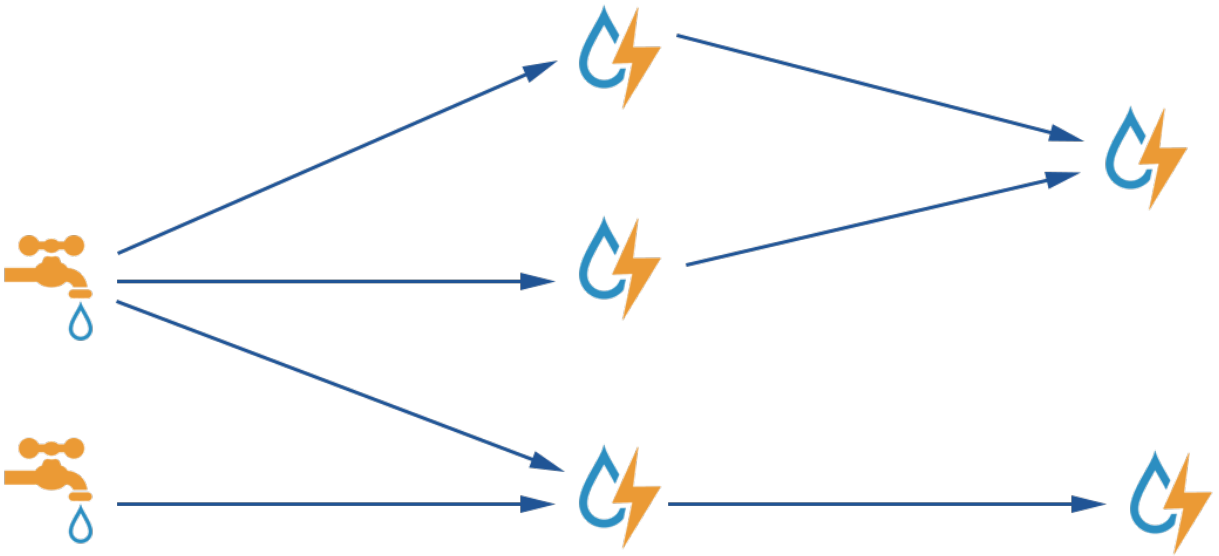
Contents and Objectives

- Storm
  - Basic Concepts
  - Quick Start
  - Other Issues

- Objectives
  - 能够针对高性能计算需求，设计并实现基于Storm的的流数据处理方案

Apache Storm

- https://storm.apache.org/index.html

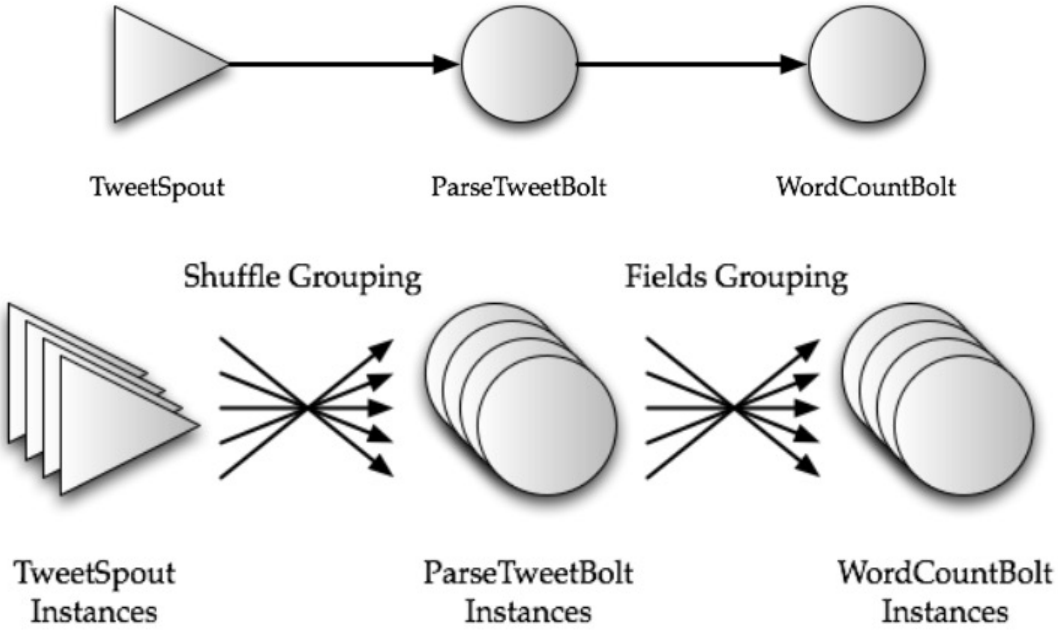


Why use Apache Storm ?

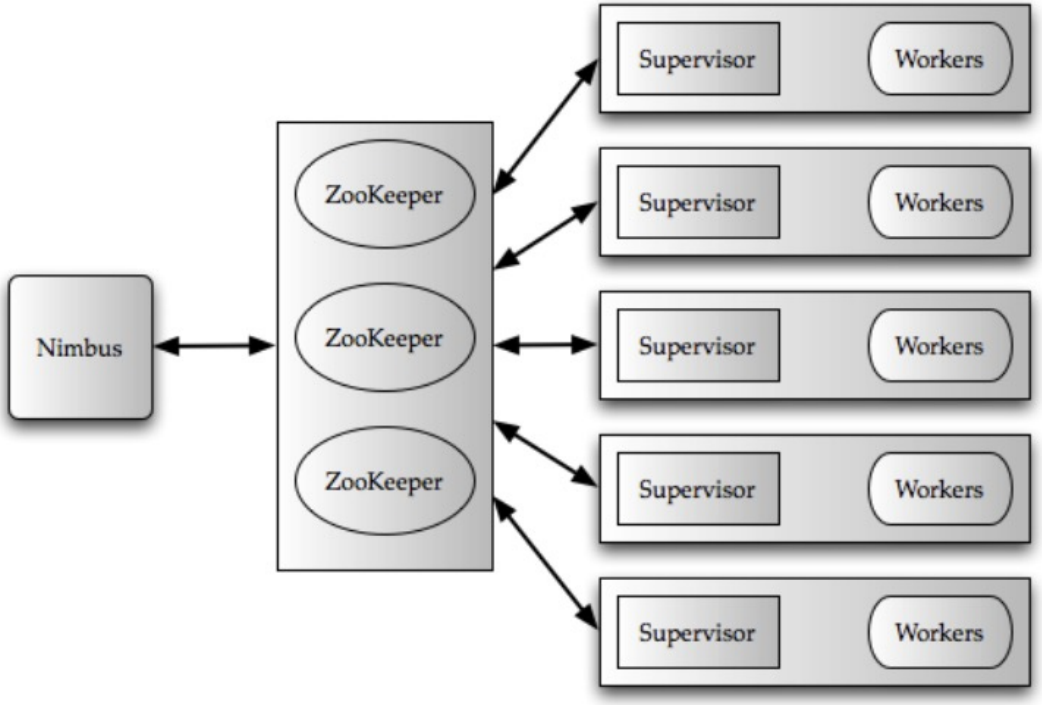
- Apache Storm is a free and open source distributed realtime computation system.
  - Apache Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing.
  - Apache Storm is simple, can be used with any programming language, and is a lot of fun to use!
- Apache Storm has many use cases:
  - realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Apache Storm is fast: a benchmark clocked it at over **a million tuples processed per second per node**. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.
- Apache Storm integrates with the queueing and database technologies you already use.
  - An Apache Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed. Read more in the tutorial

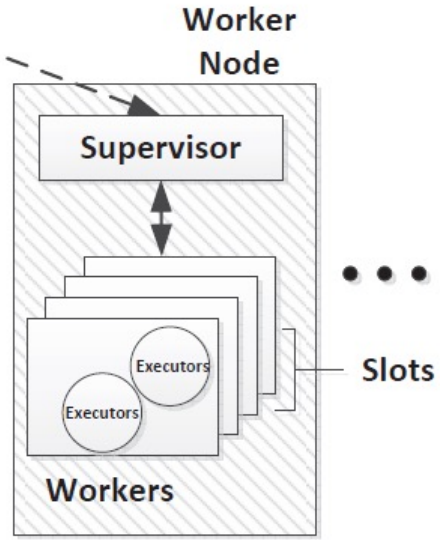
Apache Storm

- Word Count



Apache Storm Architecture





Components of a Storm Cluster

- A Storm cluster is superficially similar to a Hadoop cluster.
  - Whereas on Hadoop you run "MapReduce jobs", on Storm you run "topologies".
  - "Jobs" and "topologies" themselves are very different -- one key difference is that a MapReduce job eventually finishes, whereas a topology processes messages forever (or until you kill it).

- There are two kinds of nodes on a Storm cluster:
  - the master node and the worker nodes.
  - The master node runs a daemon called "Nimbus" that is similar to Hadoop's "JobTracker".
  - Nimbus is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures.

Components of a Storm Cluster

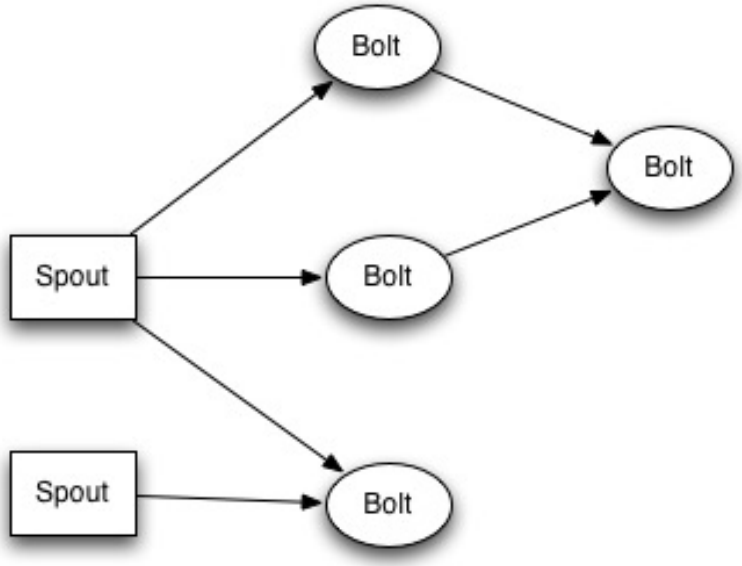
- Each worker node runs a daemon called the "Supervisor".
  - The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it.
  - Each worker process executes a subset of a topology; a running topology consists of many worker processes spread across many machines.

- All coordination between Nimbus and the Supervisors is done through a Zookeeper cluster.
  - Additionally, the Nimbus daemon and Supervisor daemons are fail-fast and stateless; all state is kept in Zookeeper or on a local disk.
  - This means you can kill -9 Nimbus or the Supervisors and they'll start back up as nothing happened. This design leads to Storm clusters being incredibly stable.

- **Topologies**
  - To do realtime computation on Storm, you create what are called “topologies”.
  - A topology is a graph of computation.
  - Each node in a topology contains processing logic, and links between nodes indicate how data should be passed around between nodes.

- **Streams**
  - The core abstraction in Storm is the “stream”.
  - A stream is an unbounded sequence of tuples.
  - Storm provides the primitives for transforming a stream into a new stream in a distributed and reliable way.

- The basic primitives Storm provides for doing stream transformations are “spouts” and “bolts”.



- The basic primitives Storm provides for doing stream transformations are “spouts” and “bolts”.
  - Spouts and bolts have interfaces that you implement to run your application-specific logic.
  - A spout is a source of streams.
    - For example, a spout may read tuples off of a Kestrel queue and emit them as a stream. Or a spout may connect to the Twitter API and emit a stream of tweets.
  - A bolt consumes any number of input streams, does some processing, and possibly emits new streams.
    - Complex stream transformations, like computing a stream of trending topics from a stream of tweets, require multiple steps and thus multiple bolts.
    - Bolts can do anything from run functions, filter tuples, do streaming aggregations, do streaming joins, talk to databases, and more.

Apache Storm

- Networks of spouts and bolts
  - are packaged into a "topology" which is the top-level abstraction that you submit to Storm clusters for execution.
  - A topology is a graph of stream transformations where each node is a spout or bolt.
  - Edges in the graph indicate which bolts are subscribing to which streams.
  - When a spout or bolt emits a tuple to a stream, it sends the tuple to every bolt that subscribed to that stream.
  - Links between nodes in your topology indicate how tuples should be passed around.
  - Each node in a Storm topology executes in parallel.
  - A topology runs forever, or until you kill it.

Data Model

- Storm uses tuples as its data model.
  - A tuple is a named list of values, and a field in a tuple can be an object of any type.
  - Out of the box, Storm supports all the primitive types, strings, and byte arrays as tuple field values.
  - To use an object of another type, you just need to implement a serializer for the type.

- Every node in a topology must declare the output fields for the tuples it emits.

Data Model

- For example,
  - this bolt declares that it emits 2-tuples with the fields "double" and "triple"

```
public class DoubleAndTripleBolt extends BaseRichBolt {
 private OutputCollectorBase _collector;

 @Override public void prepare(Map conf, TopologyContext context, OutputCollectorBase collector) {
   _collector = collector;
 }

 @Override public void execute(Tuple input) {
   int val = input.getInteger(0);
   _collector.emit(input, new Values(val*2, val*3));
   _collector.ack(input);
 }

 @Override public void declareOutputFields(OutputFieldsDeclarer declarer) {
   declarer.declare(new Fields("double", "triple"));
 }
}
```

A simple topology

- Let's look at the ExclamationTopology definition from storm-starter:

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("words", new TestWordSpout(), 10);
builder.setBolt("exclaim1", new ExclamationBolt(), 3)
        .shuffleGrouping("words");
builder.setBolt("exclaim2", new ExclamationBolt(), 2)
        .shuffleGrouping("exclaim1");
```

- This topology contains a spout and two bolts.
  - The spout emits words, and each bolt appends the string "!!!" to its input.
  - The nodes are arranged in a line: the spout emits to the first bolt which then emits to the second bolt.
  - If the spout emits the tuples ["bob"] and ["john"], then the second bolt will emit the words ["bob!!!!!!"] and ["john!!!!!!"].

A simple topology

- If you wanted component "exclaim2"
  - to read all the tuples emitted by both component "words" and component "exclaim1", you would write component "exclaim2"'s definition like this:

```
builder.setBolt("exclaim2", new ExclamationBolt(), 5)
    .shuffleGrouping("words")
    .shuffleGrouping("exclaim1");
```

A simple topology

- Spouts are responsible for emitting new messages into the topology.
  - TestWordSpout in this topology emits a random word from the list ["nathan", "mike", "jackson", "golda", "bertels"] as a 1-tuple every 100ms.
  - The implementation of nextTuple() in TestWordSpout looks like this:

```
public void nextTuple() {
  Utils.sleep(100);
  final String[] words =
          new String[] {"nathan", "mike", "jackson", "golda", "bertels"};
  final Random rand = new Random();
  final String word = words[rand.nextInt(words.length)];
  _collector.emit(new Values(word));
}
```

A simple topology

- ExclamationBolt appends the string "!!!" to its input.

```
public static class ExclamationBolt implements IRichBolt {
 OutputCollector _collector;

 @Override public void prepare(Map conf, TopologyContext context, OutputCollector collector) {
   _collector = collector;
 }

 @Override public void execute(Tuple tuple) {
   _collector.emit(tuple, new Values(tuple.getString(0) + "!!!"));
   _collector.ack(tuple);
 }

 @Override public void cleanup() { }

 @Override public void declareOutputFields(OutputFieldsDeclarer declarer) {
  declarer.declare(new Fields("word"));
 }
 @Override public Map<String, Object> getComponentConfiguration() {
  return null;
 }
}
```

A simple topology

- ExclamationBolt can be written more succinctly by extending BaseRichBolt, like so:

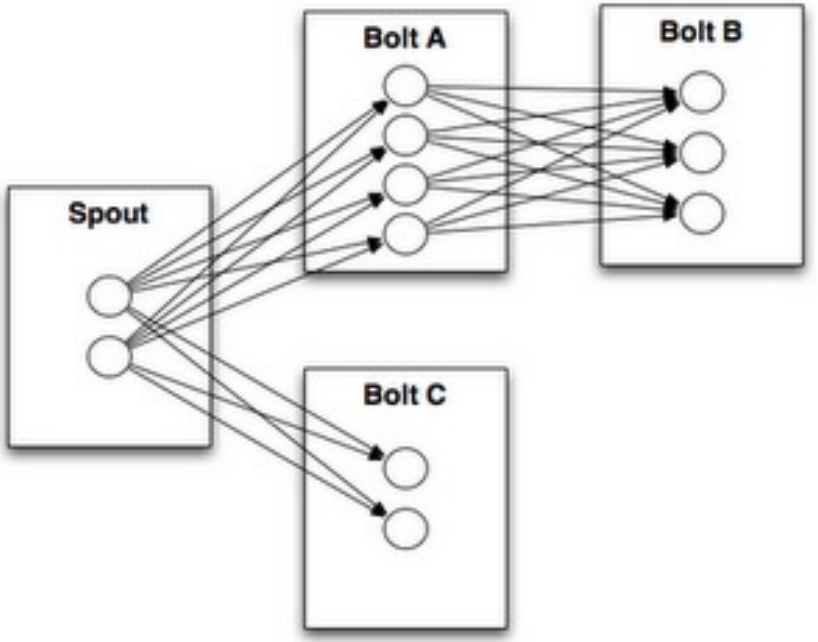
```
public static class ExclamationBolt extends BaseRichBolt {
 OutputCollector _collector;

 @Override public void prepare(Map conf, TopologyContext context, OutputCollector collector){
   _collector = collector;
 }

 @Override public void execute(Tuple tuple) {
   _collector.emit(tuple, new Values(tuple.getString(0) + "!!!"));
   _collector.ack(tuple);
 }

 @Override public void declareOutputFields(OutputFieldsDeclarer declarer) {
   declarer.declare(new Fields("word"));
 }
}
```

Running Topology in Local Mode

- Storm has two modes of operation:
  - local mode and distributed mode.
  - In local mode, Storm executes completely in a process by simulating worker nodes with threads. Local mode is useful for testing and development of topologies.

- Common configurations for local mode
  - **Config.TOPOLOGY_MAX_TASK_PARALLELISM**: This config puts a ceiling on the number of threads spawned for a single component.
  - **Config.TOPOLOGY_DEBUG**: When this is set to true, Storm will log a message every time a tuple is emitted from any spout or bolt.
  - To launch your topology in local mode you could run
  - `$ storm local topology.jar <MY_MAIN_CLASS> -c topology.debug=true`

- **Stream groupings**
  - A stream grouping tells a topology how to send tuples between two components.



- **Stream groupings**
  - Remember, spouts and bolts execute in parallel as many tasks across the cluster.
  - If you look at how a topology is executing at the task level, it looks something like this:
  - When a task for Bolt A emits a tuple to Bolt B, which task should it send the tuple to?
  - A “stream grouping” answers this question by telling Storm how to send tuples between sets of tasks.

Apache Storm Starter Examples

REliable, INtelligent & Scalable Systems

```
public class ExclamationTopology extends ConfigurableTopology {

    public static void main(String[] args) throws Exception {
        ConfigurableTopology.start(new ExclamationTopology(), args);
    }

    protected int run(String[] args) {
        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout("word", new TestWordSpout(), 10);
        builder.setBolt("exclaim1", new ExclamationBolt(), 3).shuffleGrouping("word");
        builder.setBolt("exclaim2", new ExclamationBolt(), 2).shuffleGrouping("exclaim1");

        conf.setDebug(true);

        String topologyName = "test";

        conf.setNumWorkers(3);
```

Apache Storm Starter Examples


```
        if (args != null && args.length > 0) {
            topologyName = args[0];
        }

        return submit(topologyName, conf, builder);
    }

    public static class ExclamationBolt extends BaseRichBolt {
        OutputCollector collector;

        @Override
        public void prepare(Map<String, Object> conf, TopologyContext context,
                OutputCollector collector) {
            this.collector = collector;
        }
```

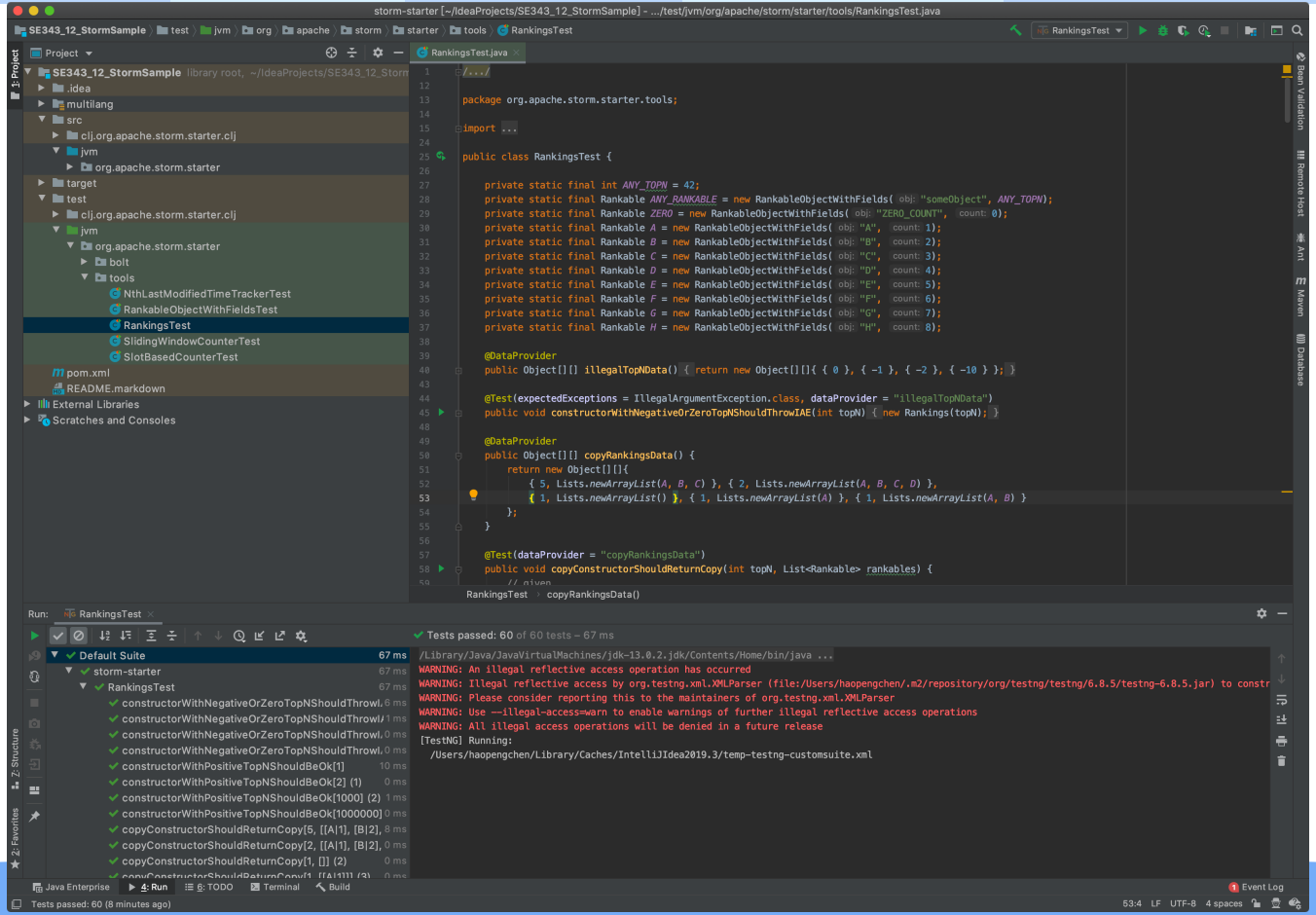
Apache Storm Starter Examples

REliable, INtelligent & Scalable Systems

```
        @Override
        public void execute(Tuple tuple) {
            collector.emit(tuple, new Values(tuple.getString(0) + "!!!"));
            collector.ack(tuple);
        }

        @Override
        public void declareOutputFields(OutputFieldsDeclarer declarer) {
            declarer.declare(new Fields("word"));
        }

    }
}
```

Apache Storm Starter Examples

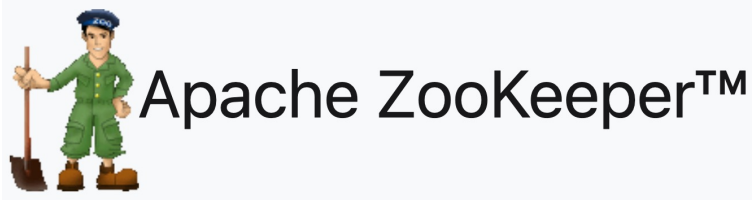
REliable, INtelligent & Scalable Systems



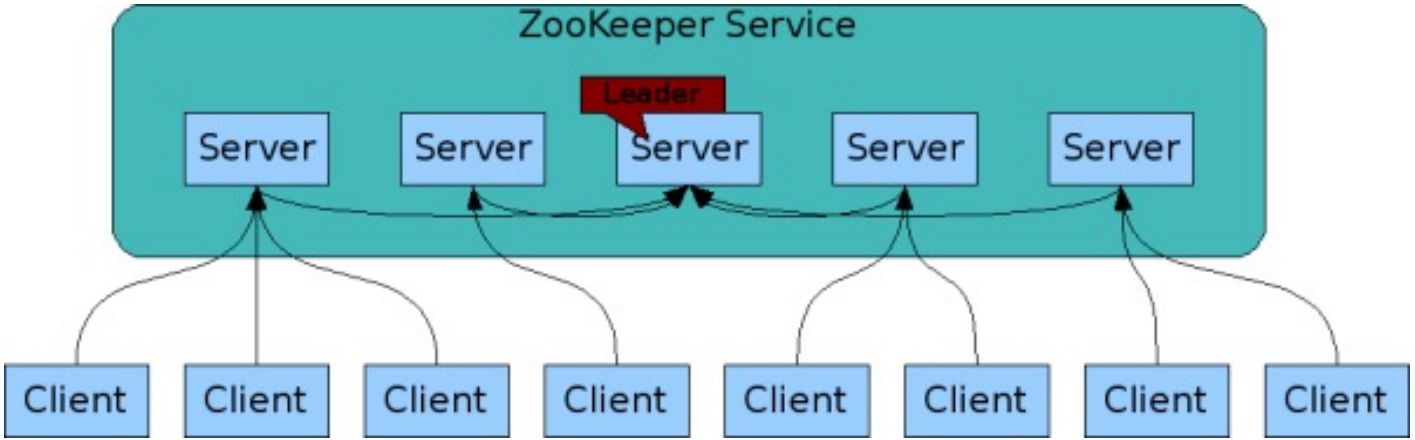
Apache ZooKeeper

- What is ZooKeeper?
  - ZooKeeper is a high-performance coordination service for distributed applications.
  - It exposes common services - such as naming, configuration management, synchronization, and group services - in a simple interface so you don't have to write them from scratch.
  - You can use it off-the-shelf to implement consensus, group management, leader election, and presence protocols. And you can build on it for your own, specific needs.

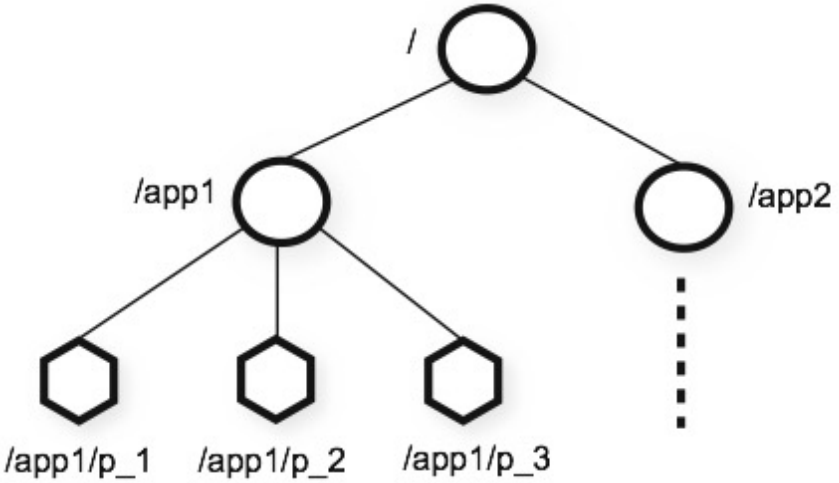
  - https://zookeeper.apache.org/



- **Design Goals**
  - ZooKeeper is simple.
  - ZooKeeper is replicated.
  - ZooKeeper is ordered.
  - ZooKeeper is fast.



- **Data model and the hierarchical namespace**
  - The namespace provided by ZooKeeper is much like that of a standard file system.
  - A name is a sequence of path elements separated by a slash (/).
  - Every node in ZooKeeper's namespace is identified by a path.
- ZooKeeper's Hierarchical Namespace



Guarantees

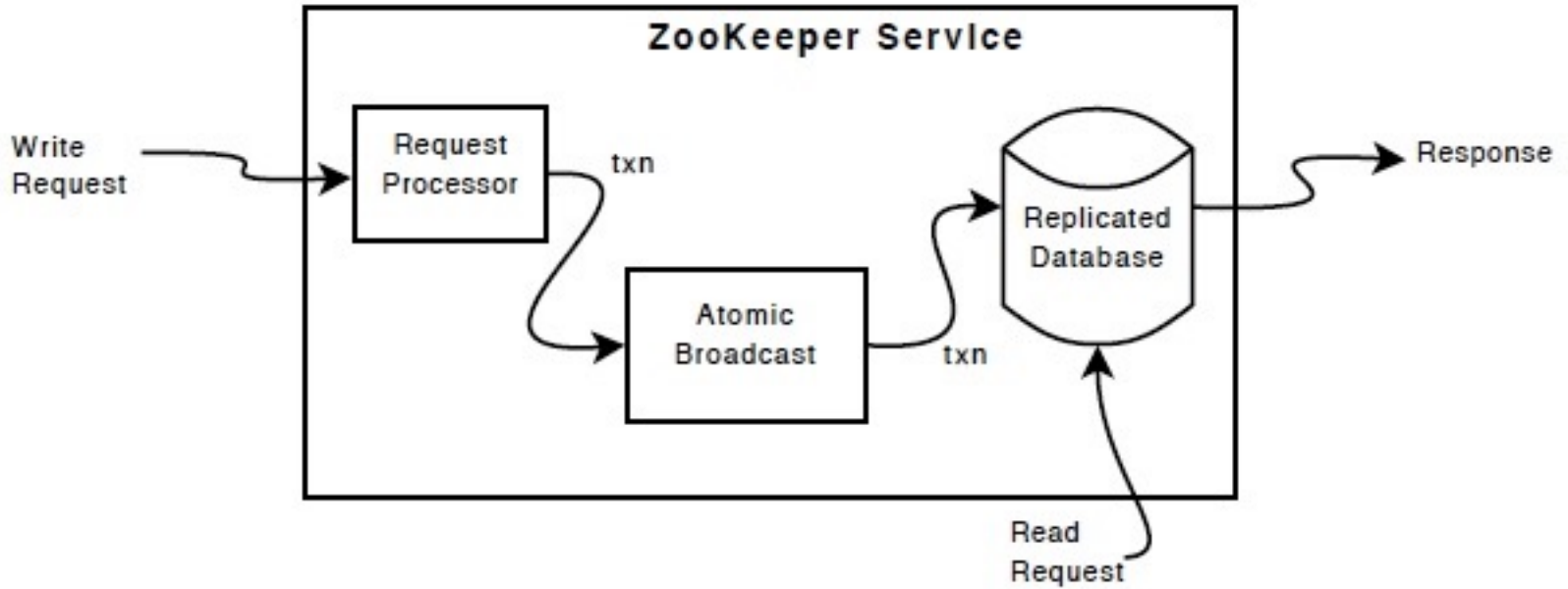
- ZooKeeper is very fast and very simple. It provides a set of guarantees. These are:
  - Sequential Consistency - Updates from a client will be applied in the order that they were sent.
  - Atomicity - Updates either succeed or fail. No partial results.
  - Single System Image - A client will see the same view of the service regardless of the server that it connects to. i.e., a client will never see an older view of the system even if the client fails over to a different server with the same session.
  - Reliability - Once an update has been applied, it will persist from that time forward until a client overwrites the update.
  - Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

Simple API

- One of the design goals of ZooKeeper is providing a very simple programming interface. As a result, it supports only these operations:
  - *create* : creates a node at a location in the tree
  - *delete* : deletes a node
  - *exists* : tests if a node exists at a location
  - *get data* : reads the data from a node
  - *set data* : writes data to a node
  - *get children* : retrieves a list of children of a node
  - *sync* : waits for data to be propagated

Implementation

- ZooKeeper Components shows the high-level components of the ZooKeeper service.



Standalone Operations

- To start ZooKeeper you need a configuration file. Here is a sample, create it in **conf/zoo.cfg**:

```
tickTime=2000
dataDir=/var/zookeeper
clientPort=2181
```

- Here are the meanings for each of the fields:
  - ***tickTime*** : the basic time unit in milliseconds used by ZooKeeper. It is used to do heartbeats and the minimum session timeout will be twice the tickTime.
  - ***dataDir*** : the location to store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database.
  - ***clientPort*** : the port to listen for client connections
- Now that you created the configuration file, you can start ZooKeeper:

```
bin/zkServer.sh start
```

Standalone Operations

- **Connecting to ZooKeeper**

  $ bin/zkCli.sh -server 127.0.0.1:2181

- This lets you perform simple, file-like operations.

- Once you have connected, you should see something like:

  Connecting to localhost:2181
  log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
  log4j:WARN Please initialize the log4j system properly.
  Welcome to ZooKeeper!
  JLine support is enabled
  [zkshell: 0]

Start Apache Storm

- The Storm release contains a file at **conf/storm.yaml**

```
storm.zookeeper.servers:
  - "127.0.0.1"
nimbus.seeds: [" 127.0.0.1 "]
storm.local.dir: $STORM_HOME/storm-local
supervisor.slots.ports:
  - 6700
  - 6701
  - 6702
  - 6703
```

- Here's how to run the Storm daemons:
  - **Nimbus**: Run the command bin/storm nimbus under supervision on the master machine.
  - **Supervisor**: Run the command bin/storm supervisor under supervision on each worker machine.

References

- Apache Storm
  - http://storm.incubator.apache.org/documentation/Tutorial.html
- Example Storm Topologies
  - https://github.com/apache/storm/tree/2.3.x-branch/examples/storm-starter
- ZooKeeper Getting Started Guide
  - https://zookeeper.apache.org/doc/r3.7.0
- Setting up a Storm Cluster
  - https://storm.apache.org/releases/2.3.0/Setting-up-a-Storm-cluster.html
- Storm Tutorial
  - https://storm.apache.org/releases/2.3.0/Tutorial.html
- **Intellij IDEA 导入或运行流式处理框架storm以及java.lang.NoClassDefFoundError报错的解决方案**
  - https://blog.csdn.net/weixin_35757704/article/details/75348270
- ZooKeeper: Because Coordinating Distributed Systems is a Zoo
  - https://zookeeper.apache.org/doc/current/index.html

Thank You!