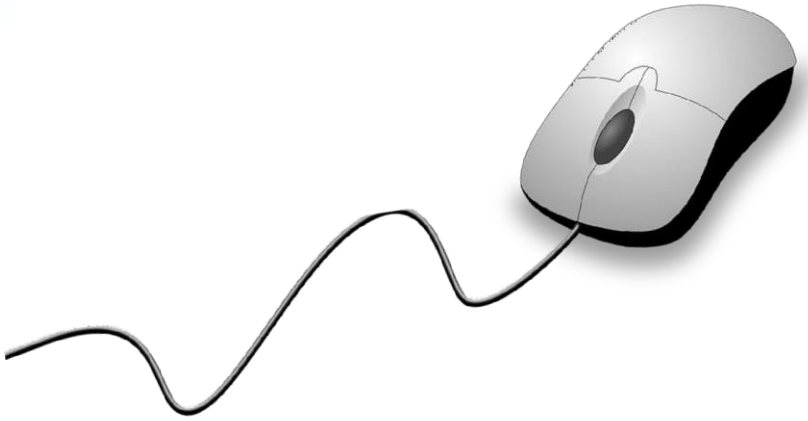
공개SW 솔루션 설치 & 활용 가이드

미들웨어> 분산시스템SW



APACHE STORM™

제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide

OPENUP

오픈소스 소프트웨어 통합지원센터

Open Source Software Support Center

CONTENTS

1. 개요

소개	• Apache Storm은 트위터의 실시간 분석과 최적화, 안티 스팸 등에 활용하다가 오픈소스로 공개했으며 실시간으로 대용량 데이터를 분석 할 수 있게 해주는 서비스		
주요기능	• 지속적인 메시지를 처리하거나 실시간으로 계산 및 DB 업데이트 등 스트리밍 프로세싱과 지속적 처리 가능		
대분류	• 미들웨어	**소분류**	• 분산시스템SW
라이선스 형태	• Apache License 2.0	**사전설치 솔루션**	• Java 1.6 이상 • Python 2.6.6 이상 • Zookeeper, ZeroMQ, JZMQ
운영체제	• Cross-platform	**버전**	• 1.2.2 (2018년 10월 기준)
특징	• 다양한 프로그래밍 언어 지원(클로저, 자바, 루비, 파이썬, Storm communication protoccol) • 데이터 처리 보장 • 장애를 자동으로 관리 • 집중적인 쿼리를 병렬처리		
보안취약점	• 취약점 ID : CVE-2018-1332 • 심각도 : 6.5 MEDIUM(V3) • 취약점 설명 : 사용자가 일부 Storm 데몬과 통신할때 다른사용자로가장할 수 있는 취약점을노출 • 대응방안 : 1.2.2 이상으로 업그레이드 • 참고 경로 : https://lists.apache.org/thread.html/50f1d6a7af27f49d2e498a9ab2975685302cd8ca47000b7c38f339a4@%3Cdev.storm.apache.org%3E		
개발회사	• BackTpye		
공식 홈페이지	• https://storm.apache.org		

2. 기능요약

- Fault-tolerant
  - Storm은 노드의 장애를 자동 관리
- 다양한 프로그래밍 언어 지원
  - Storm은 기본적으로 JVM(Java Virtual Machine) 위에서 동작하지만 Twitter의 Thrift 프로토콜을기반으로 하기 때문에 다양한 언어로 구현 가능
- Scalable
  - 멀티플 스레드, 프로세스, 서버를 이용해 병렬 처리가 가능하며 추가 확장 쉬움
- Guarantees no data loss
  - Storm은 모든 메시지가 처리 될 것을 보장하며, 유실 없이 최소한 한 번 메세지가 처리될 수 있게 지원

3. 실행환경

- 설치 요구 사항
  - Linux 또는 Mac
  - JAVA 1.6 이상
  - Python 2.6.6 이상
  - Zookeeper, ZeroMQ, JZMQ

  * Storm을 직접 빌드하기 위해서는 gcc, maven 등 각종 라이브러리가 추가 필요

4. 설치 및 실행

세부 목차

1. Storm 개발환경 설치
   1. JDK 설치 및 환경변수 설정
   2. Zookeeper 설치
   3. Maven 설치
2. Apache Storm 설치 및 기동

4. 설치 및 실행

4.1.1 JDK 설치 및 환경변수 설정

- JDK 설치
  - # yum install java-1.8.0-openjdk-devel.x86_64
- JDK 환경변수 설정
- javac 위치 확인
  - # which javac
  - # readlink -f /usr/bin/javac

```
[root@localhost tuser]# yum install java-1.8.0-openjdk-devel.x86_64
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.navercorp.com
 * epel: mirror.premi.st
 * extras: mirror.navercorp.com
 * ius: archive.linux.duke.edu
 * updates: mirror.navercorp.com
Resolving Dependencies
--> Running transaction check
---> Package java-1.8.0-openjdk-devel.x86_64 1:1.8.0.191.b12-0.el7_5 will be ins
talled
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                    Arch      Version                    Repository
                                                                           Size
================================================================================
Installing:
 java-1.8.0-openjdk-devel   x86_64    1:1.8.0.191.b12-0.el7_5    updates   9.8 M

Transaction Summary
```

```
[tuser@localhost ~]$ which javac
/usr/bin/javac
[tuser@localhost ~]$ readlink -f /usr/bin/javac
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-0.el7_5.x86_64/bin/javac
[tuser@localhost ~]$
```

  - /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-0.el7_5.x86_64 가 JAVA_HOME 될 경로
  - vi /etc/profile
- 파일 하단에 아래 내용을 추가한 뒤 저장
  - export JAVA_HOME= /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-0.el7_5.x86_64
  - export PATH=$PATH:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.191.b12-0.el7_5.x86_64/bin
  - export CLASSPATH="."
  - # source /etc/profile

4. 설치 및 실행

4.1.2 Zookeeper 설치

- Zookeeper 설치 및 실행
  - # wget https://archive.apache.org/dist/zookeeper/zookeeper-3.4.13/zookeeper-3.4.13.tar.gz
  - # tar -xvzf zookeeper-3.4.13.tar.gz
- 환경 변수 설정 파일을 생성, 샘플 파일을 편집하여 사용 가능
  - # cd zookeeper-3.4.13/conf
  - # cp zoo_sample.cfg zoo.cfg
  - # vi zoo.cfg
- Zookeeper 실행 스크립트를 이용하여 Zookeeper 실행
  - # cd ../bin
  - # ./zkServer.sh start

```
[root@localhost tuser]# cd zookeeper-3.4.12/conf
[root@localhost conf]# cp zoo_sample.cfg zoo.cfg
[root@localhost conf]# vi zoo.cfg
[root@localhost conf]# cd ../bin

[root@localhost bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /home/tuser/zookeeper-3.4.12/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@localhost bin]#
```

4. 설치 및 실행

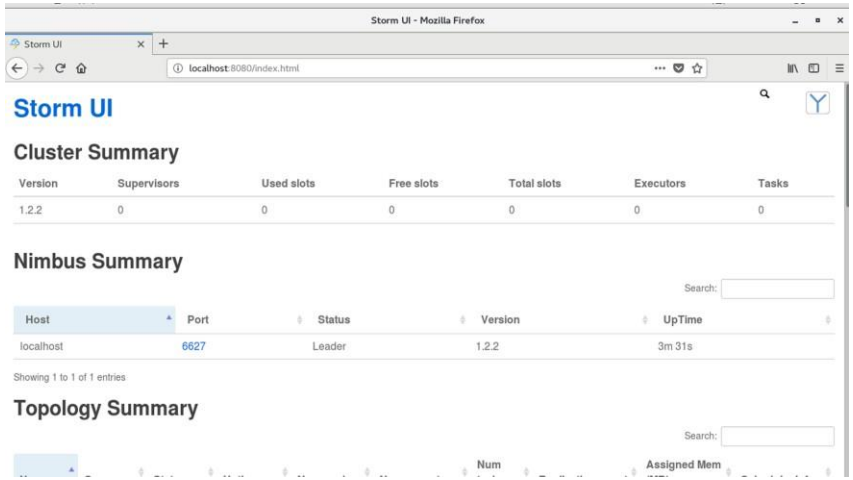
4.1.3 Maven 설치

- Maven 설치
  - # wget http://mirror.apache-kr.org/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz
  - # tar -xvzf apache-maven-3.5.4-bin.tar.gz
  - #cd ~
  - # vi /etc/profile
- 아래 내용 저장
  - export MAVEN_HOME=/home/tuser/apache-maven-3.5.4
  - export PATH=$PATH:/home/tuser/apache-maven-3.5.4/bin
  - # source /etc/profile
- 설치 확인
  - # mvn –version

4. 설치 및 실행

4.2 Apache Storm 설치 및기동

- Apache Storm 설치 및 기동
  - # wget https://archive.apache.org/dist/storm/apache-storm-1.2.2/apache-storm-1.2.2.tar.gz
  - # tar -xvzf apache-storm-1.2.2.tar.gz
  - # cd ~
- # vi /etc/profile
- 아래 내용 저장
  - export STORM_HOME=/home/tuser/apache-storm-1.2.2
  - export PATH=$PATH:/home/tuser/apache-storm-1.2.2/bin
  - # source /etc/profile
  - # cd /home/tuser/apache-storm-1.2.2
- 마스터 노드 실행
  - # bin/storm nimbus
- # bin/storm ui
- 워커노드 실행
  - # bin/storm supervisor
- Localhost:8080 으로 접속 확인 가능



5. 기능소개

세부 목차

1. storm의 동작방식
2. storm의 컴포넌트
3. 스트림 그룹핑
4. 메시지 처리방식

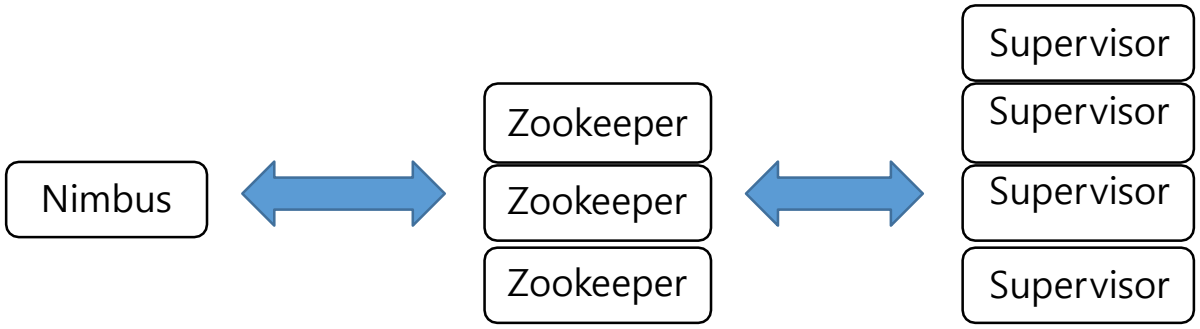
5. 기능소개

5.1 storm의 동작방식

- Storm의 아키텍처는 Hadoop과 매우 유사하며, Hadoop에서는 MR(Map-Reduce) 작업을 실행하는 반면 Storm에서는 토폴로지 작업을 수행하는 것이 다름
- MR 작업과 토폴로지의 차이

  - MR 작업은 정해진 데이터 세트를 처리한 후 완료되지만 토폴로지는 계속해서 메시지를 처리한다는 점 다름
- Storm의 클러스터는 마스터노드(Nimbus)와 워커노드(Supervisor)로 구성되며 Zookeeper 이용 관리

- Nimbus(마스터노드) : Hadoop의 JobTracker와 유사한 개념으로 클러스터 주변에 코드를 배포하고 작업을 컴퓨터에 할당하며 실패 모니터링
  - Supervisor(워커노드) : 실제적으로 워커 프로세스의 시작과 종료, 실행 상태 모니터링 등 수행
  - Zookeeper : 분산된 노드 간의 관리를 수행하고 시스템의 안정성을 유지하도록 관리 역할



5. 기능소개

5.2 storm의 컴포넌트(1/3)

- Stream : storm의 중요한 추상 개념, 데이터가 끊임없이 연속적으로 들어올 때 각각의 데이터를 병렬 분산해 튜플이라는 형태로 관리

  * 튜플 : 정렬된 목록

- Spout : 스파우트는 데이터가 스톰 토폴로지로 들어가는 입구이며, 어댑터로써 동작하는데 데이터 소스와 연결을 맺고 데이터를 튜플로 변환하여 스트림으로 튜플을 내보내는 일을 하며, 스파우트의 4가지 중요한 메소드

- open() : Spout가 처음 초기화 될 때 한번만 호출되는 메소드로 데이터 소스로부터의 연결을 초기화 등의 역할 수행
- nextTuple() : 데이터 스트림 하나를 읽고나서 다음 데이터 스트림을 읽을때 호출되는 메소드
- ack(Object msgId) : 데이터 스트림이 성공적으로 처리되었을때 호출되는데성공처리된메세지를지우는등 성공 처리에 대한 후처리 구현
- fail(Object msgId) : 해당 데이터 스트림이 Storm Topology를 수행하던 중에 에러가발생하거나타임아웃 등이 발생 했을때 호출되는데 이 때 사용자가 에러데 대한 에러 처리 로직 명시

5. 기능소개

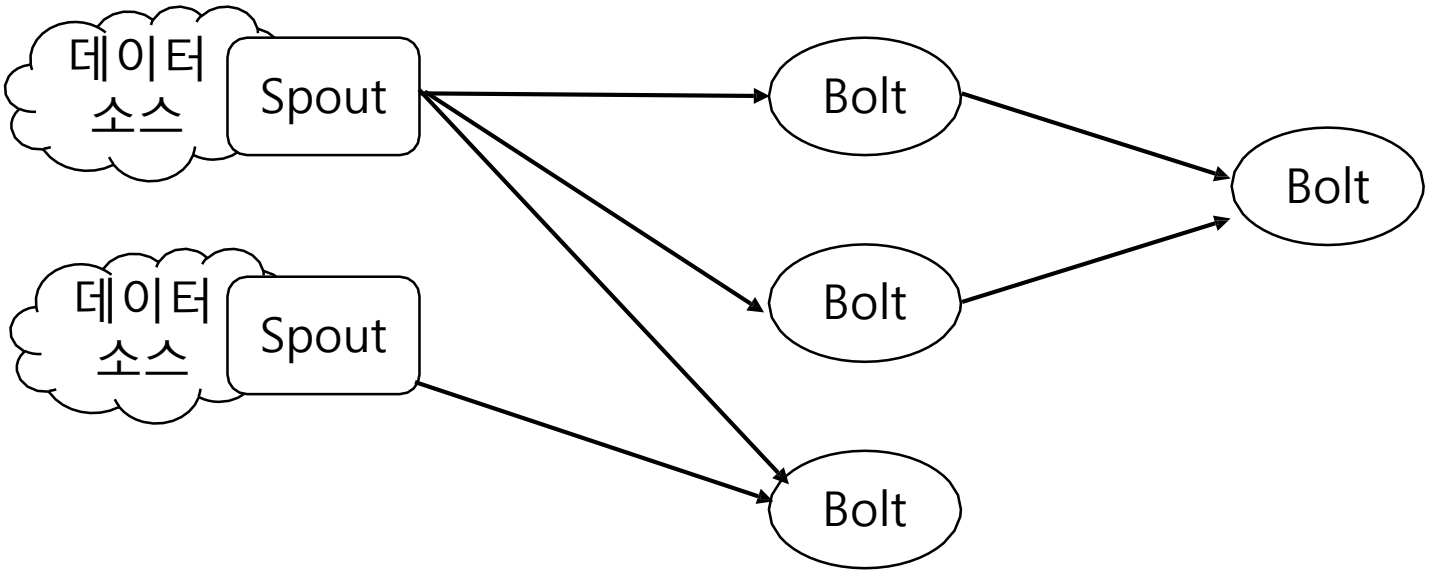
5.2 storm의 컴포넌트(2/3)

- Bolt : 볼트는 실시간 연산의 연산자나 함수로 생각할 수 있으며, 다수의 스트림을 입력 받아 데이터를 처리하고 선택적으로 하나 이상의 스트림으로 내보내며, 볼트가 처리하는 일반적인 기능은 튜플 필터링, 조인과 집계, 연산, 데이터베이스 읽기/쓰기
  - execute(Tuple input) : 가장 필수적인 메소드로 Bolt에 들어온 메세지를 처리하는 로직을 갖고, 종단 Bolt가 아닌 경우에는 다음 Bolt로 메세지 전달

5. 기능소개

5.2 storm의 컴포넌트(3/3)

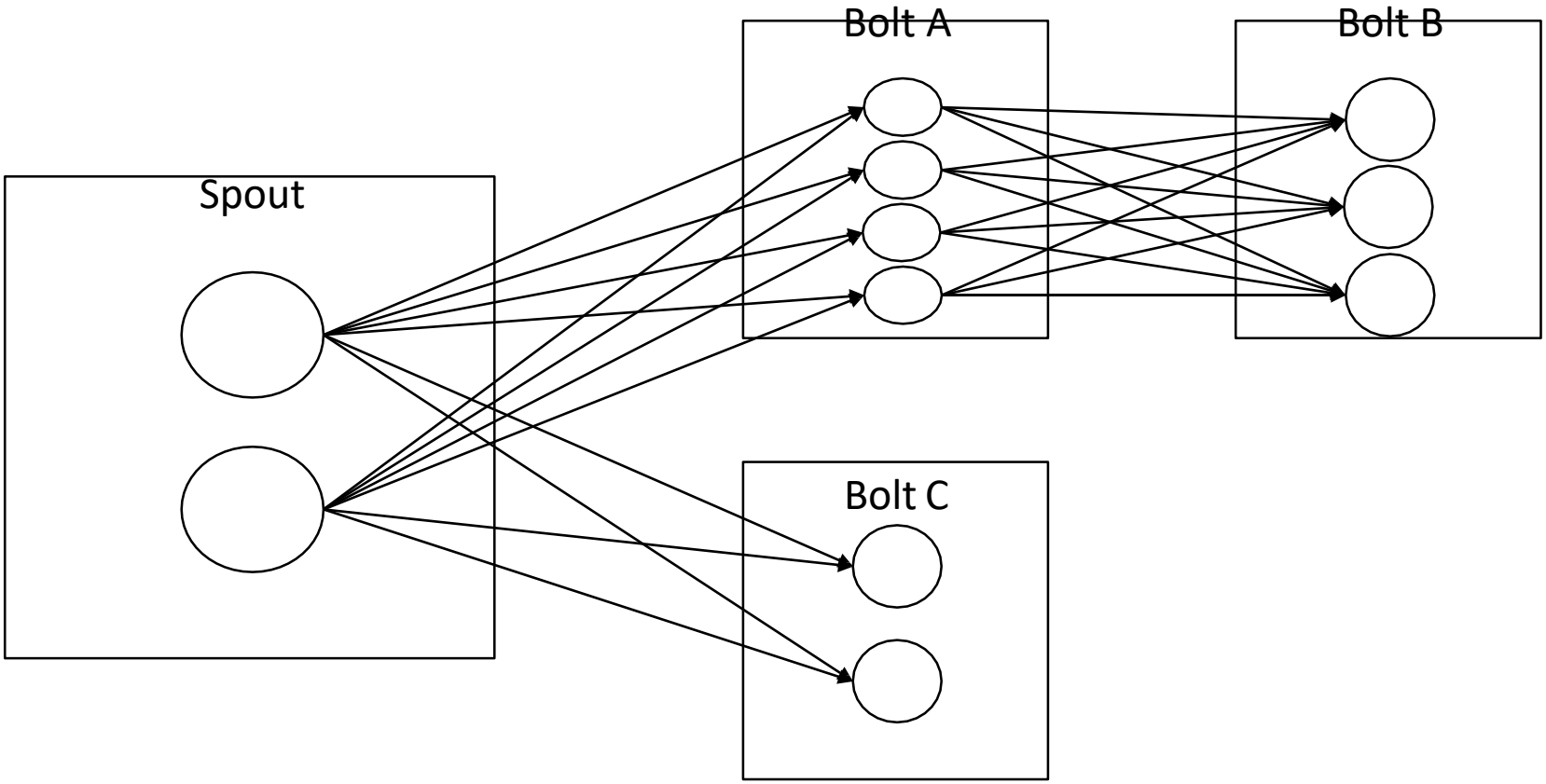
- Topology : 토폴로지라고 부르는 스톰의 분산 연산 구조는 데이터 스트림, 스파우트, 볼트로 구성되며, 스톰 토폴로지는 하둡과 같은 배치 처리 시스템의 잡과 거의 비슷하지만 배치 잡은 연산의 처음과 끝지점이 명확하게 정의되어 있는 반면 스톰 토폴로지는 죽이거나 언디플로이(undeploy)할 때까지 계속 동작함



5. 기능소개

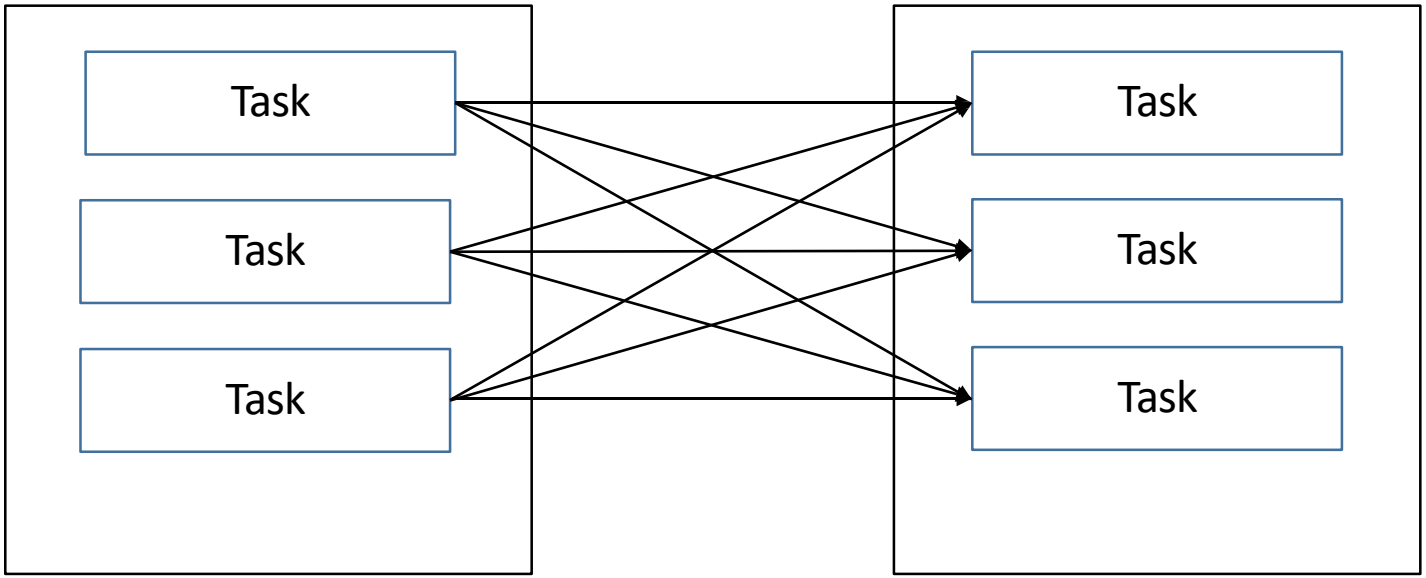
5.3 스트림 그룹핑(1/6)

- 스트림 그룹핑은 스트림의 튜플들이 다수의 볼트로 어떻게 분산되는지에 대해 정의한 것이며, 스톰은 7가지의 스트림 그룹핑 제공



< 스트림의 전달경로 >

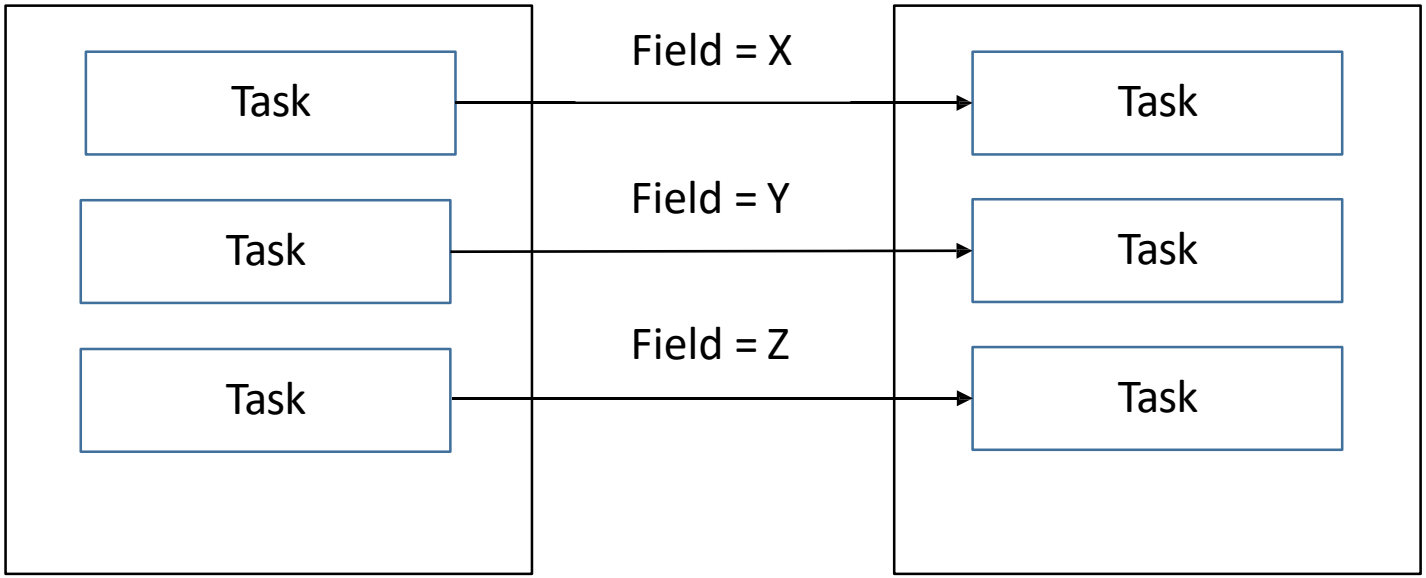
- 셔플 그룹핑(Shuffle grouping) : 튜플을 무작위로 동일한 비율로 나눠서 볼트의 task에 할당하며, 기본적인 분산 처리 방식임



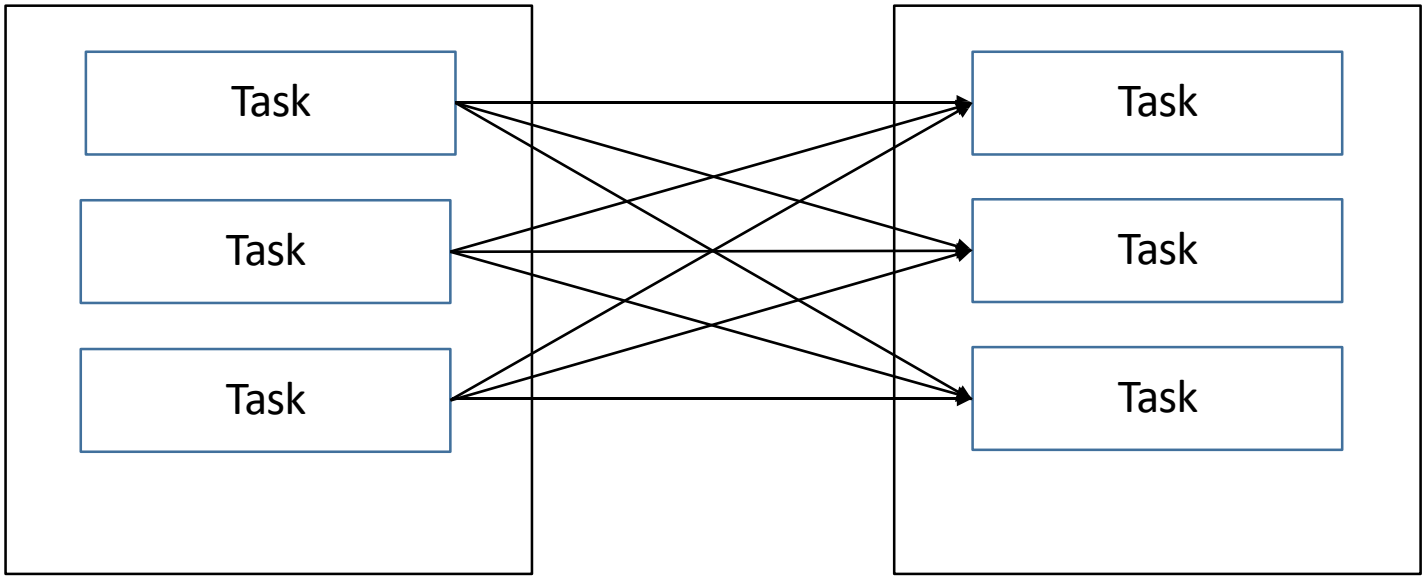
5. 기능소개

5.3 스트림 그룹핑(3/6)

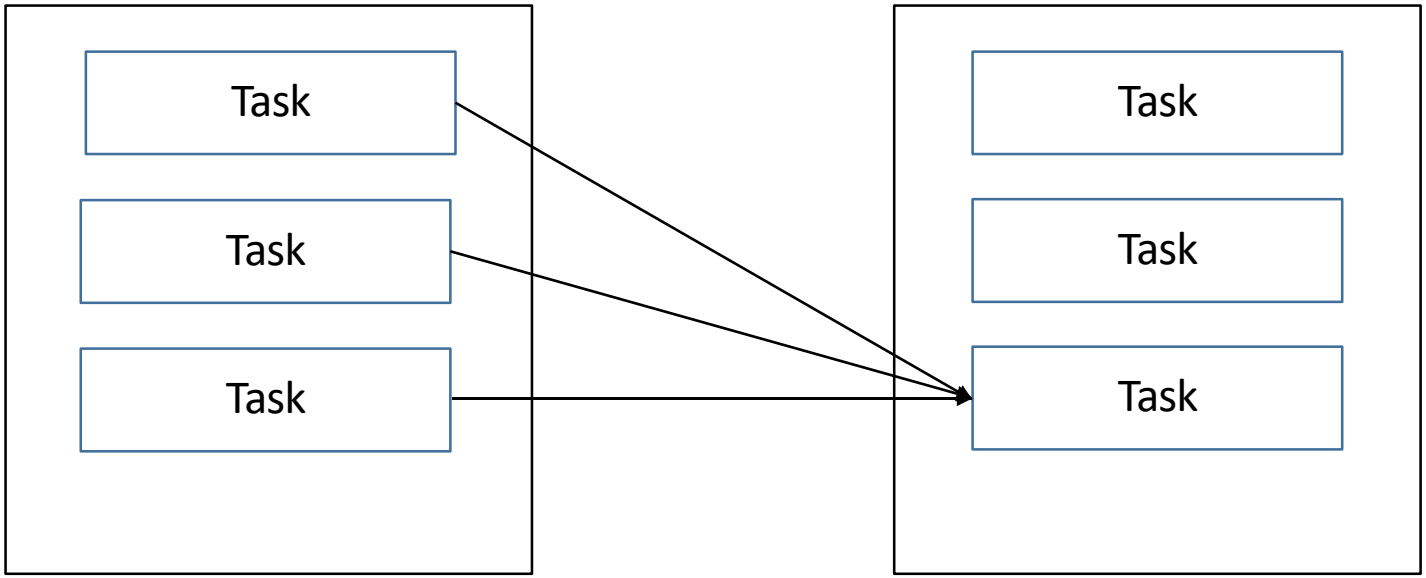
- 필드 그룹핑(Field grouping) : 그룹핑에서 정의한 필드의 값에 따라 튜플이 전달될 볼트가 정해짐



- 올 그룹핑(All grouping) : 튜플 스트림의 복사본들이 모든 볼트 작업단위로 전달되며, 각각의 작업단위는 튜플의 복사본을 받음

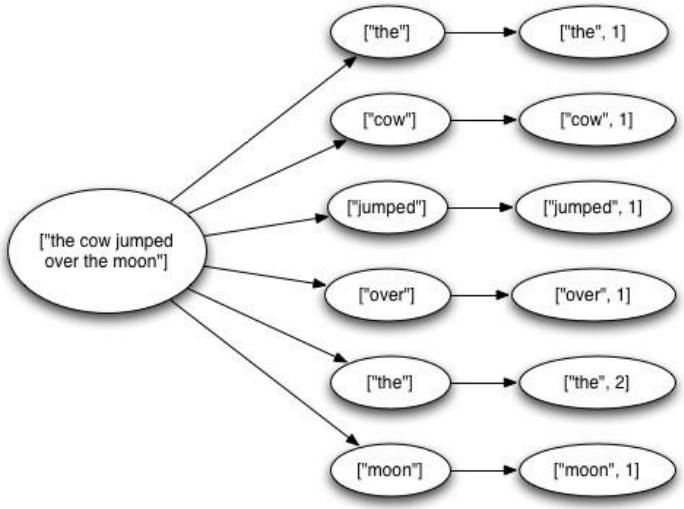


- 글로벌 그룹핑(Global grouping) : 스트림의 모든 튜플을 단일 작업단위로 전달하며, 이 때 작업단위 아이디값이 가장 작은 작업단위로 전달
  - 글로벌 그룹핑을 사용하는 볼트의 병렬화 지수를 늘리거나 작업단위의 수를 늘리는 것은 아무런 의미 없는 행동임



- 넌 그룹핑(None grouping) : 넌 그룹핑은 기능적으로 셔플 그룹핑과 동일함
- 다이렉트 그룹핑(Direct grouping) : 소스 스트림이 emitDirect() 메소드를 호출하여 튜플을 내보내고 싶은 컴포넌트를 직접 지정 가능 하며, 다이렉트 스트림으로 선언된 스트림에서만 사용 가능
- 로컬 또는 셔플 그룹핑(Local or shuffle grouping) : 로컬 또는 셔플 그룹핑은 셔플 그룹핑과 비슷하지만 같은 워커 프로세스에서 동작하는 볼트 작업단위 대상으로만 튜플을 셔플링하며, 그 외 기능은 셔플 그룹핑과 동일함
  - 토폴로지의 병렬화 지수에 따라 로컬 또는 셔플 그룹핑은 토폴로지의 성능을 네트워크 전송 한계까지 올릴 수 있음

- Storm은 스파우트에서 생성된 메시지가 모두 처리되도록 보장하는 기능 제공하며, 스파우트에서 생성된 튜플마다 이 튜플을 최상위 노드로 하는 튜플 트리를 생성하고 볼트가 이 튜플을 처리하면서 생기는 튜플을 이 튜플 트리에 하위 노드로 덧붙임
  - Storm은 이 튜플 트리의 모든 노드를 토폴로지당 설정된 시간 안에 처리했는지 혹은 볼트가명시적으로 실패 메시지를 보내 이 튜플의 처리에 실패했는지 여부 감지
  - 스파우트에 이 사실을 알려 스파우트가 해당 튜플에 대한 처리를 재시도할지 말지를 결정하도록 함
  - 이때 설정된 시간은 TOPOLOGY_MESSAGE_TIMEOUT_SECS 파라미터(기본값은 30초)로 변경 가능



* 튜플 트리 참조 : http://storm.apache.org/releases/1.0.6/Guaranteeing-message-processing.html >

6. 활용예제

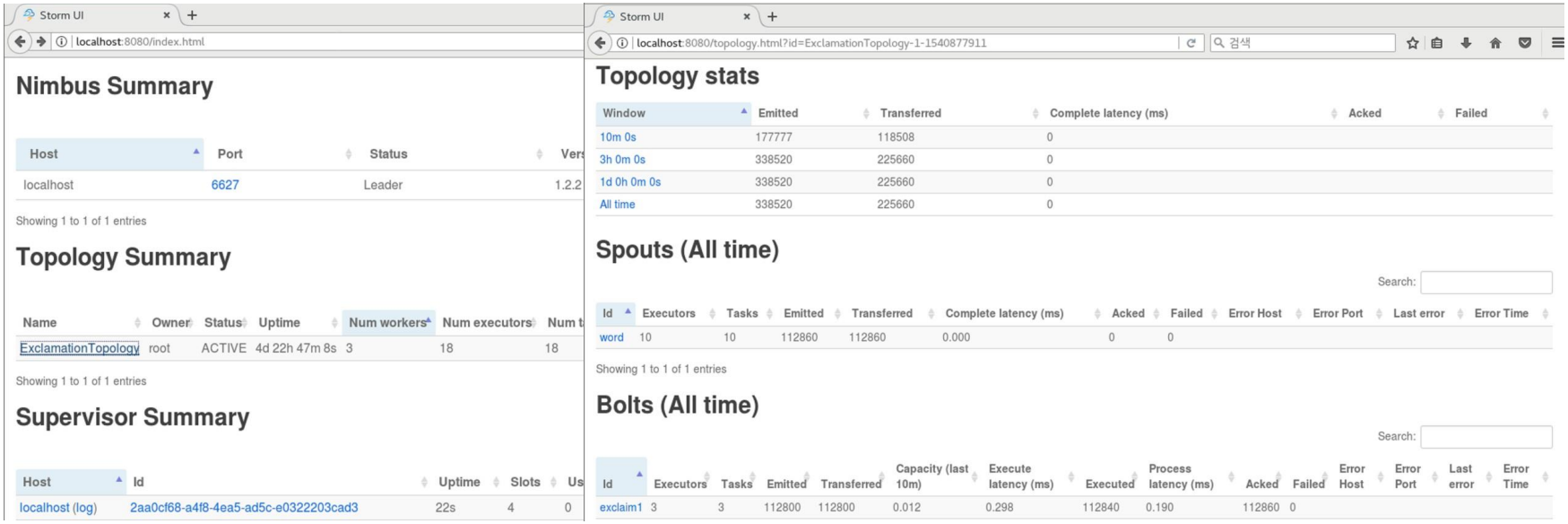
세부 목차

1. 샘플 프로그램 테스트
2. 샘플 프로그램 설명

6. 활용예제

6.1 샘플 프로그램 테스트

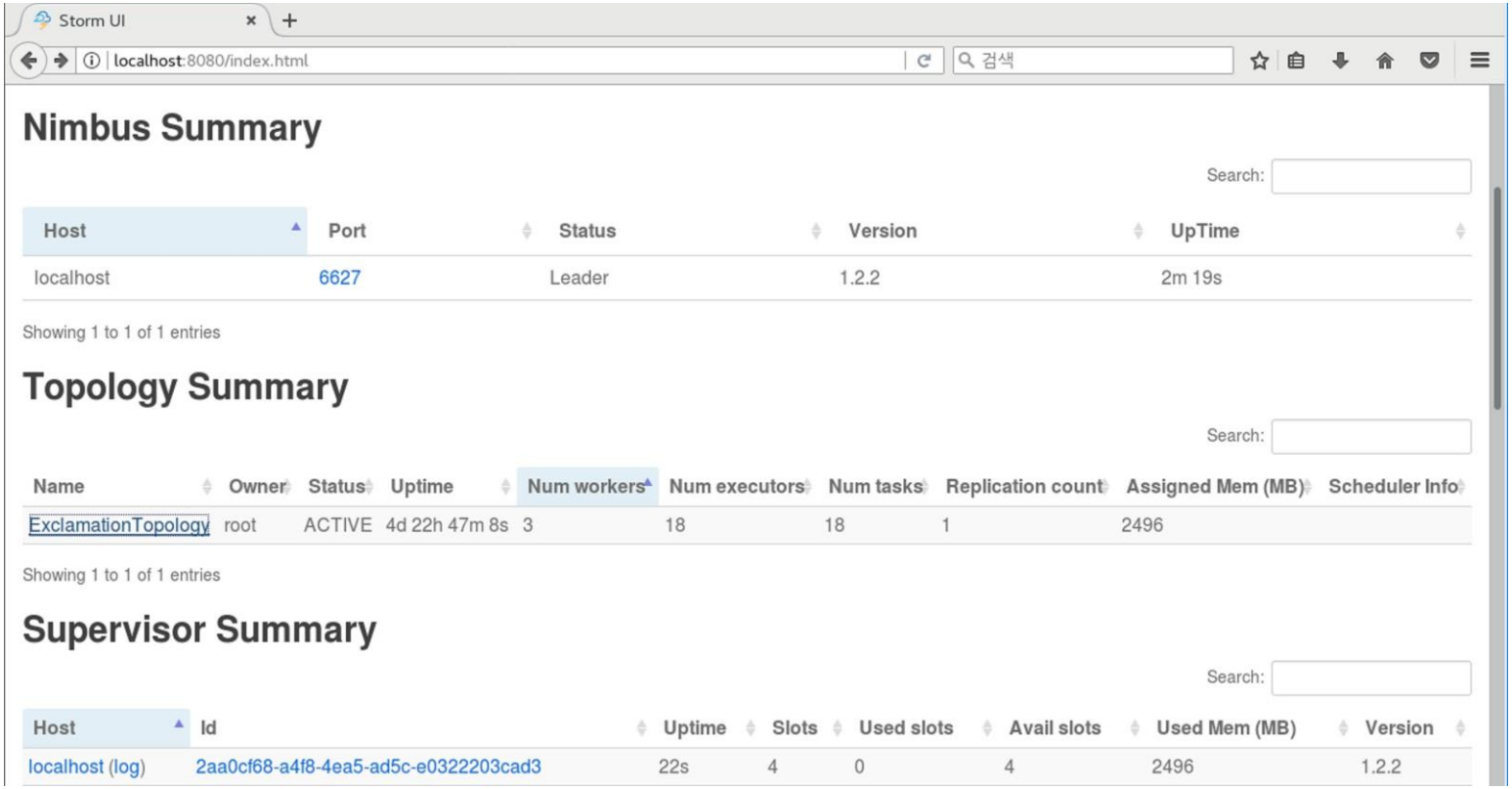
- 샘플 프로그램
  - # cd /home/tuser/apache-storm-1.2.2/examples/storm-starter
  - # mvn clean install –DskipTests=true
  - # mvn package
  - # storm jar target/storm-starter-1.2.2.jar org.apache.storm.starter.ExclamationTopology



6. 활용예제

6.2 샘플프로그램 설명(1/8)

- ExclamationTopology 를 Topology에 배포 중
- 토폴로지의 이름을 클릭하면 토폴로지의 상태 정보를 확인 가능



6. 활용예제

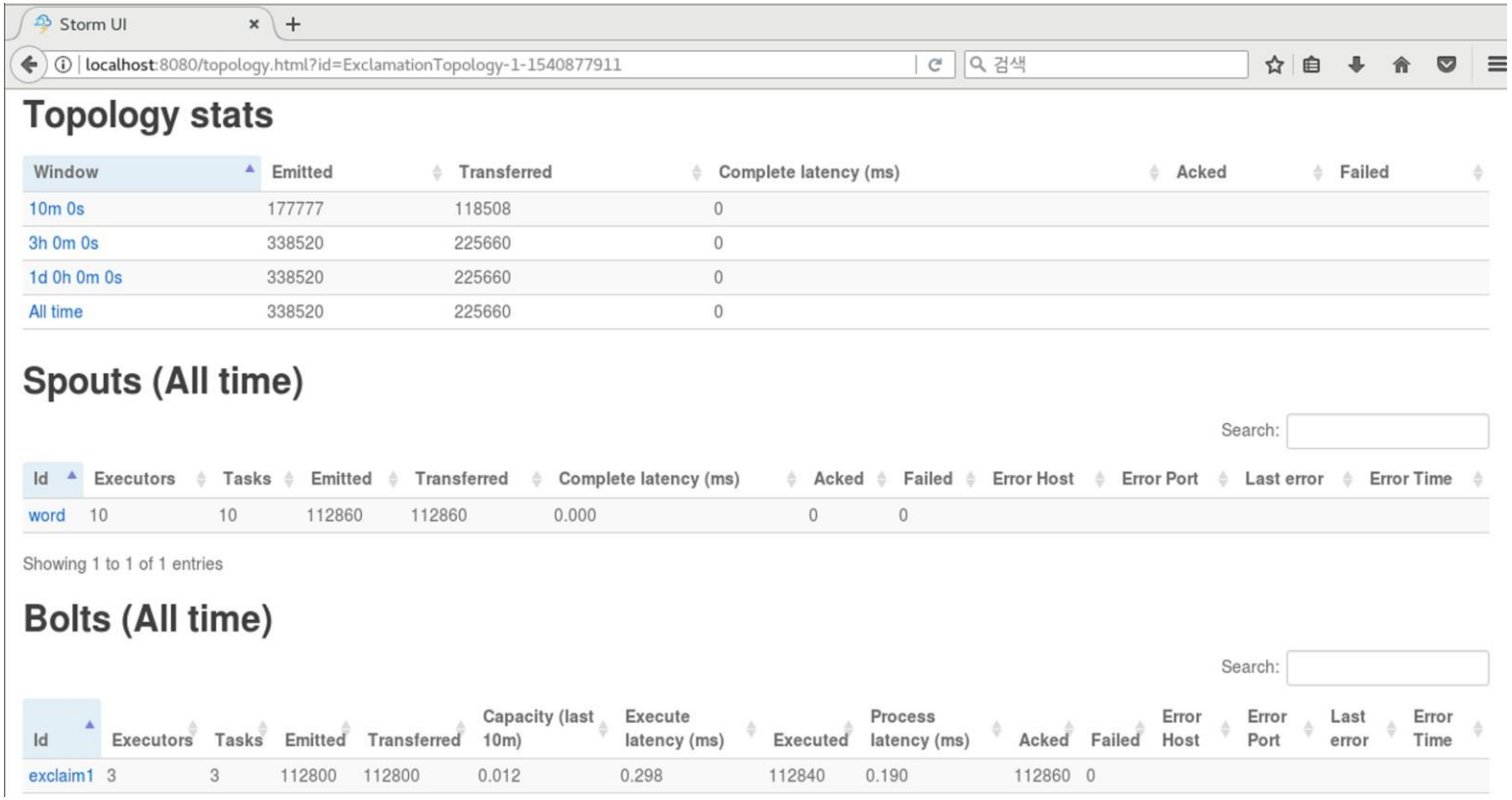
6.2 샘플프로그램 설명(2/8)

- Topology Summary
  - Name : 토폴로지가 배포되었을 때 토폴로지에 부여된 이름
  - Id : 토폴로지가 시작될 때마다 토폴로지에 부여되는 고유 ID
  - Status : 상태는 active, inactive, killed 또는 rebalancing 중 하나일 수 있음
  - Uptime : 토폴로지가 배포된 이후의 시간
  - Num workers : 현재 토폴로지에서 사용되는 작업자 수
  - Num executors : 현재 토폴로지에서 사용되는 executor 수
  - Num tasks : 현재 토폴로지에서 사용되는 작업 수
- Supervisor summary
  - Id : 클러스터에 조인할 때 Supervisor에게 부여되는 고유 식별자
  - Host : 원격 호스트에서 보고한 호스트 이름
  - Uptime : Supervisor 가 클러스터에 등록된 시간
  - Slots : 대상 호스트의 작업자 수
  - Used slots : 해당 호스트에서 사용된 작업자 수

6. 활용예제

6.2 샘플프로그램 설명(3/8)

- Window : 통계가 적용되는 과거 기간
- Emitted : 방출된 튜플 수
- Transferred : 하나 이상의 볼트로 전송된 튜플 수



- Complete latency : 토폴로지에서 튜플 "트리" 를 완전히 처리하는데 걸리는 평균 시간
- Acked : 성공적으로 처리된 튜플 "트리" 수
- Failed : 작업이 완료되기 전에 실패했거나 시간 초과된 튜플 "트리"의 수

• Spouts

- Id : 토폴로지에서 구성 요소에 할당한 ID
- Executor : 스파우트에 할당된 실행 프로그램의 수
- Tasks : 스파우트에 할당된 작업 수
- Transferred : 전송 된 튜플의 수

6. 활용예제

6.2 샘플프로그램 설명(5/8)

- Bolts
  - Id : 토폴로지에서 구성 요소에 할당한 ID
  - Executors : 볼트에 할당된 실행 프로그램의 수
  - Tasks : 볼트에 할당된 작업 수
  - Emitted : 볼트에서 배출된 튜플 수
  - Transferred : 볼트에서 전송된 튜플 수
  - Execute latency : 튜플이 execute 메소드에서 보낸 평균 시간
  - Executed : 처리된 들어오는 튜플의 수
  - Process latency : 튜플이 처음 수신된 후 ack하는데 걸리는 평균 시간
  - Acked : 볼트가 허락한 튜플 수
  - Failed : 볼트가 실패한 튜플 수

6. 활용예제

6.2 샘플프로그램 설명(6/8)

- /examples/storm-starter/src/jvm/org/apache/storm/starter/ExclamationTopology.java 를 확인하면 해당 Topology에서는 다음과 같이 Bolt 정의

```
public class ExclamationTopology {

  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"));
    }


  }
```

```
  public static void main(String[] args) throws Exception {
    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");

    Config conf = new Config();
    conf.setDebug(true);

    if (args != null && args.length > 0) {
      conf.setNumWorkers(3);

      StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
    }
    else {

      LocalCluster cluster = new LocalCluster();
      cluster.submitTopology("test", conf, builder.createTopology());
      Utils.sleep(10000);
      cluster.killTopology("test");
      cluster.shutdown();
    }
  }
}
```

* declareOutputFields() : ExclamationBolt가 word라는 하나의 필드를 내 보내도록 선언

6. 활용예제

6.2 샘플프로그램 설명(7/8)

- TopologyBuilder builder = new TopologyBuilder();
  - Topology를 Builder를 통해 만듦
- builder.setSpout("word", new TestWordSpout(), 10);
  - TestWordSpout이라는 Spout Class에 word라는 이름을 붙여 할당
- builder.setBolt("exclaim1", new ExclamationBolt(), 3).shuffleGrouping("word");
  - 위에서 설정한 words spout을 shuffleGrouping이라는 방식으로 ExclamationBolt라는 Bolt Class로 보내고 이름은 exclaim1으로 설정
- builder.setBolt("exclaim2", new ExclamationBolt(), 2).shuffleGrouping("exclaim1");
  - 위에서 설정한 exclaim1을 shuffleGrouping하여 이번에도 똑같이 ExclamationBolt로 보내고 그 이름을 exclaim2으로 설정

6. 활용예제

6.2 샘플프로그램 설명(8/8)

- prepare(): bolt로부터 내보내지는 튜플들에 사용되는 OutputCollector라는 형식을 미리 준비
- excute(): bolt로 들어온 튜플을 한 개씩 받아서 excute()내의 로직 처리, ExclamationBolt에 excute() 내에서 첫 번째 라인은 받아온 튜플에 "!!!"를 붙여 내보냄(emit), 두 번째 라인은 입력 받은 튜플 ack함, 이것은 스톰의 데이터 손실을 막기 위한 방법
- declareOutputFields() : ExclamationBolt가 "word"라는 하나의 필드를 내 보내도록 선언

```
public class ExclamationTopology {

  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"));
    }


  }
```

7. FAQ

Q **Apache Storm에 SSL이 포함되어 있습니까?**

A 포함 되어있지 않으며, Apache Storm에는 보안 socket layer가 없습니다. 법적 또는 관료적인 문제를 피하기 위해 SSL은 Apache Storm에 포함되어 있지 않습니다.

Q **Apache Kafka와 Apache Storm의 차이점은 무엇입니까?**

A 실제 구현 후에만 전문가가 이해할 수 있는 두 가지 사소한 차이가 있습니다. Apache Kafka는 많은 양의 데이터를 처리 할 수 있는 분산 된 강력한 메시징 시스템으로 한쪽 끝에서 다른 쪽 끝으로 메시지를 전달합니다. 동시에 Apache Storm은 필요한 실시간 조작 및 데이터 계산을 수행하는 분산 실시간 컴퓨팅 시스템입니다. 요약하면 Apache Storm은 Kafka 모듈에서 데이터를 가져와서 필요한 조작을 수행합니다.

8. 용어정리

용어	설 명
튜플(Tuple)	실시간으로 생성되어 전달되는 데이터 구조체
스트림(Stream)	튜플들은 비동기적으로 이것을 처리하는 로봇에게 던져지는데 이 튜플의 흐름
스파우트(Spout)	Stream의 데이터 소스, 하나의 Spout은 하나 이상의 Stream을 발생시킬 수 있으며, Kestrel, RabbitMQ, Kafka 등을 통해 외부로부터 데이터를 받아오는 역할
볼트(Bolt)	튜플들의 스트림을 전달받아 이를 처리하는 로봇
님버스(Nimbus)	데몬이 마스터 노드 역할
수퍼바이저(Supervisor)	Graphical User Interface
CLI	실제적으로 워커 프로세스의 시작과 종료, 실행 상태 모니터링 등을 수행

Open Source Software Installation & Application Guide