Storm

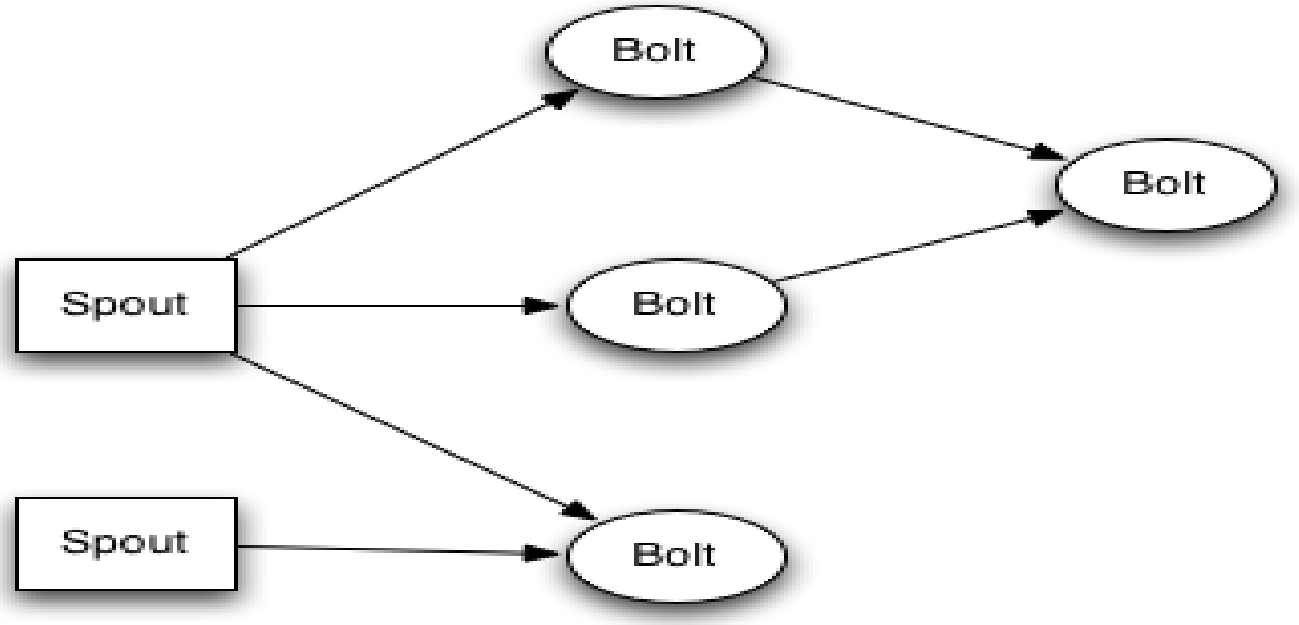
Система realtime параллельной обработки данных

Зачем нужен Storm ?

- Системы batch procesing (Hadoop и т.д.) предоставляют данные с большим временным интервалом
- Очереди (queue messaging) добавляют в систему дополнительную “большую коробку” (big box)
- Сложность масштабирования и конфигурации систем использующих очереди
- Низкий уровень абстракции messaging систем

Идея Storm

- Описываем на простом DSL топологию прохождения данных про графу
- Запускаем топологию в кластере Storm



Архитектура Storm

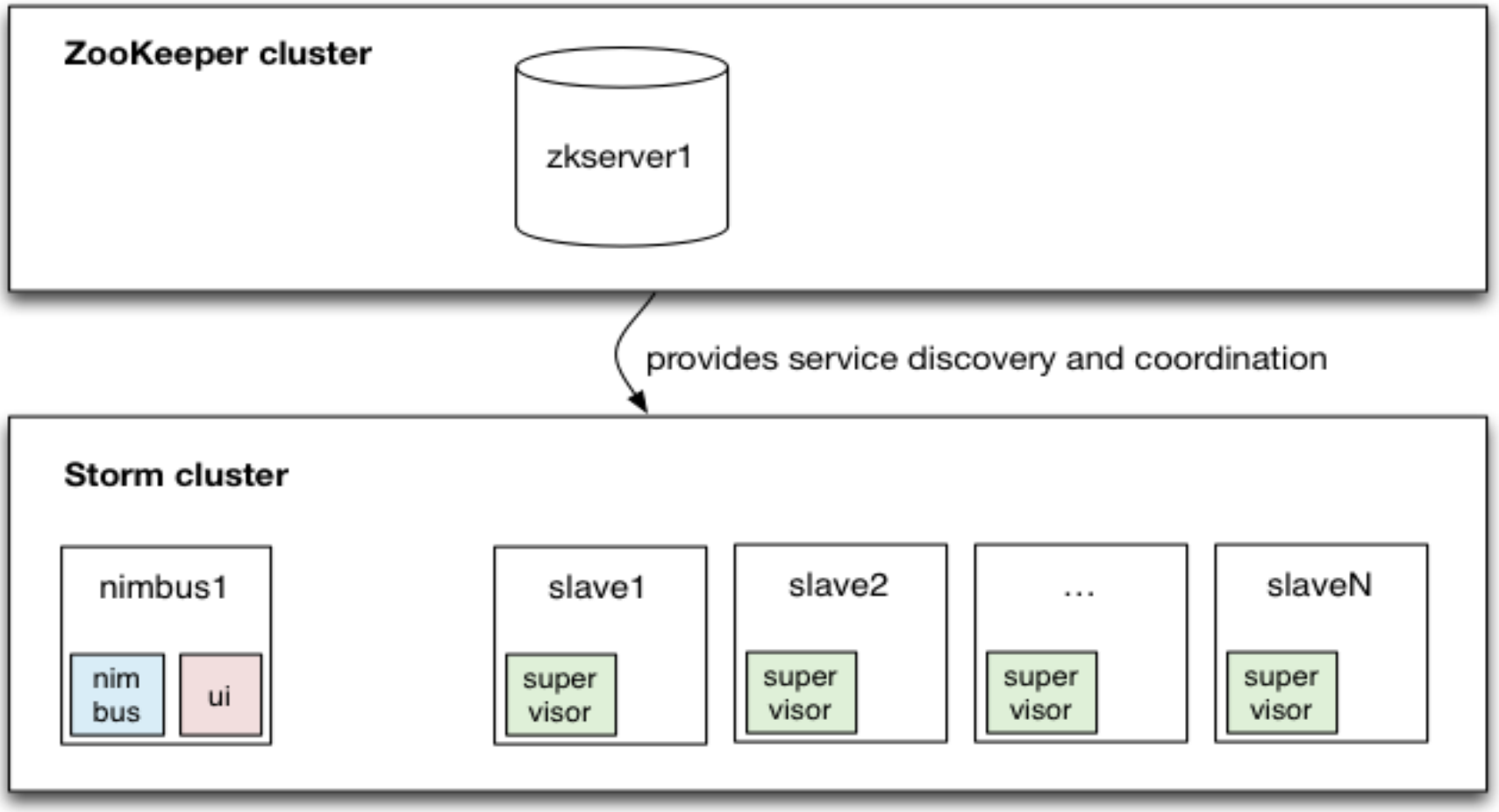
- В Storm используется 3 категории узлов :
- Nimbus – аналог JobTracker

  Распределяет задачи (tasks) по узлам, запускает и останавливает топологии, обрабатывает ошибки
- Узлы Zookeeper

  координирующий сервис,хранит конфигурацию, отслеживает состояние узлов
- Supervisor

  Получает от nimbus задания, запускает выполнения tasks топологии

Архитектура Storm



Ключевые понятия Storm

- Storm – система обработки потока сообщений, проходящего через граф.
- Каждый узел производит обработку входных данных и генерацию данных для следующего узла
- Единица потока данных – сообщение (tuple)

  Набор примитивов, имеющих порядок и имена
- Узел Spout

  Только генерирует сообщения(обычно связан со внешней системой)
- Узел Bolt

  Получает сообщения на вход и генерирует новые

Spout

- Генерирует tuple в бесконечном цикле
- Релизует IrichSpout
- Ключевые методы :
  - declareOutputFields, описывает исходящие tuple
  - open, инициализирует spout
  - nextTuple, генерирует следующее сообщение
    - NextTuple вызывается в бесконечном цикле

Пример минимального Spout

```
public class SimpleSpout extends BaseRichSpout {
    private SpoutOutputCollector spoutOutputCollector;
    private int count = 0;
    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        outputFieldsDeclarer.declare(new Fields("line"));
    }
    @Override
    public void open(Map conf, TopologyContext topologyContext, SpoutOutputCollector spoutOutputCollector) {
        this.spoutOutputCollector = spoutOutputCollector;
    }
    @Override
    public void nextTuple() {
        if(count<10) {
            spoutOutputCollector.emit(Lists.<Object>newArrayList("test" + count));
            count++;
        }
    }
}
```

Bolt

- Принимает tuple, обрабатывает его и (необязательно) генерирует исходящие tuple
- Реализует IrichBolt
- Ключевые методы :
  - declareOutputFields, описывает исходящие tuple
  - execute, обрабатывает входящий tuple и (возможно) генерирует исходящие

Пример минимального Bolt

```
public class SimpleReceiveBolt extends BaseRichBolt {
   private OutputCollector outputCollector;
    @Override
    public void prepare(Map map, TopologyContext topologyContext, OutputCollector outputCollector) {
        this.outputCollector = outputCollector;
    }
    @Override
    public void execute(Tuple tuple) {
        System.out.println("receive tuple="+tuple);
        outputCollector.ack(tuple);
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
    }
}
```

Создание топологии

- Используется класс TopologyBuilder
- setSpout добавляет в топологию Spout
- setBolt добавляет болт и привязывает его к предыдущему узлу
- Пример:

```
TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("generator",new SimpleSpout());
        builder.setBolt("printer", new SimpleReceiveBolt())
        .shuffleGrouping("generator");
```

Группировка tuple

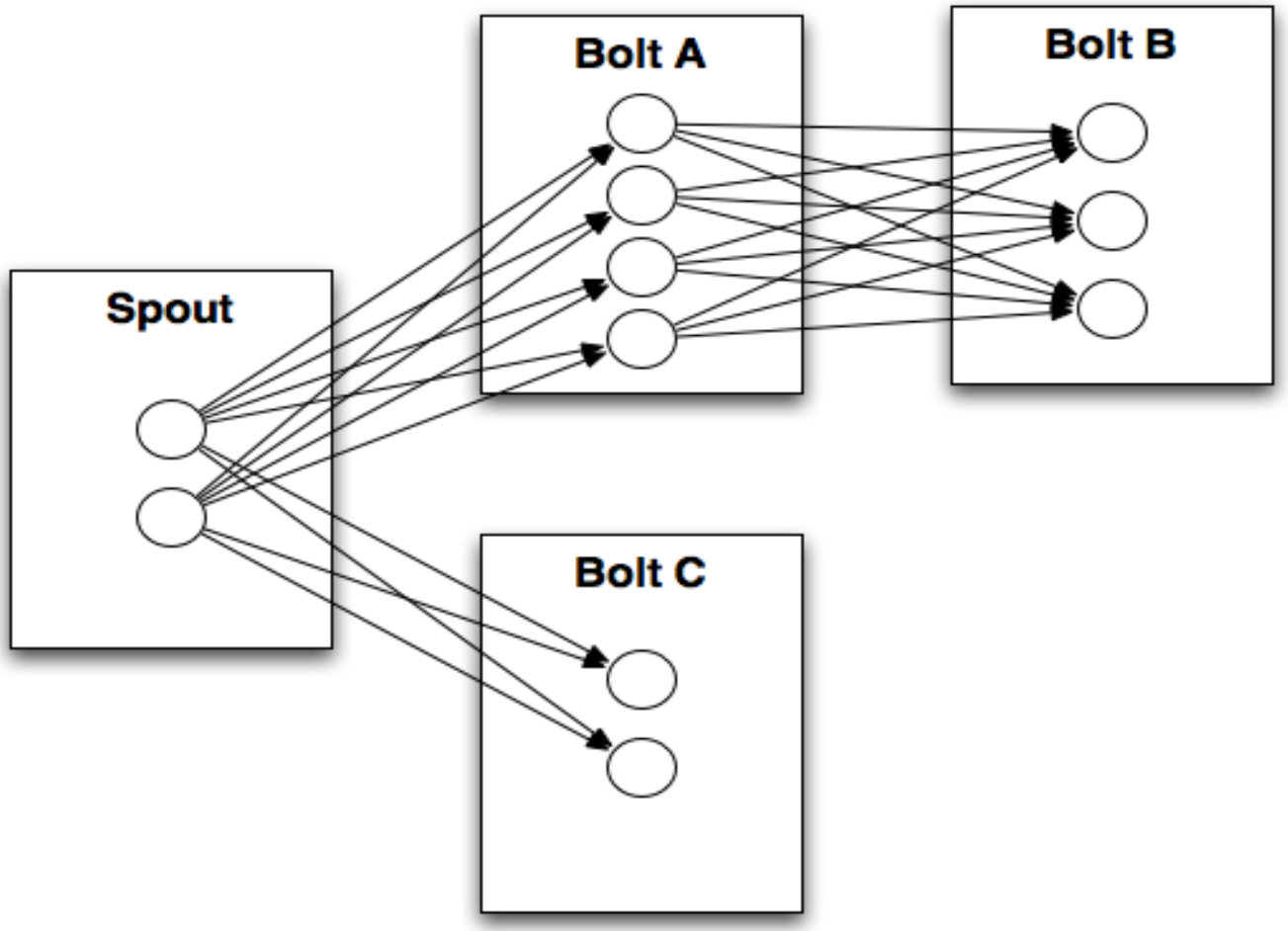
- Аналог партиционирования в Hadoop
- Группировка определяет алгоритм распределения tuple по принимающим узлам
- Задается в момент связывания узлов при создании топологии
- Экземпляров одного узла может быть много
- Пример:
  - builder.setBolt("printer", new SimpleReceiveBolt())
    .shuffleGrouping("generator");

Поток данных

- Bolt и Spout в методе declareOutputFields могут задекларировать несколько потоков исходящих данных
- Для каждого потока задается набор полей
  - outputFieldsDeclarer.declareStream("words", new Fields("line"));
  - outputFieldsDeclarer.declareStream("sync", new Fields("command"));
- В момент связывания узлов можно явно указать поток данных

```
builder.setBolt("splitter", new SimpleSplitBolt(), 10).shuffleGrouping("generator",
"words");
```

Пример работы группировки



Виды группировки

- Shuffle Grouping,
  - bolt выбирается случайно, равномерно используются все экземаляры
- Fields Grouping
  - Для группировки указывается ряд полей. Гарантируется что записи с одинаковым набором значений попадут одному и тому экземпляру bolt
- All Grouping
  - Сообщение отсылается каждому экземпляру bolt
-

Виды группировки

- Custom Grouping
  - Реализуем интерфейс CustomStreamGrouping и сами управляем группировкой
- Direct Grouping
  - В момент посылки сообщения bolt или spout (посылающий сообщение) явно указывает номер bolt приемника
- Global Grouping
  - Все сообщения уходят одному bolt (имеющему минимальный номер)

Подтверждение сообщений

- Надежные (reliable) bolt и spout посылают вместе с сообщением id сообщения
- Принимающий bolt может подтвердить успешную обработку(ack) или послать сообщение об ошибке(fail)
- Отправитель получает нотификацию о каждом сообщении которое он отправил

API acknowledge

- Методы для работы с подтверженими

```
public interface ISpout extends Serializable {
...
    void ack(Object msgId);
    void fail(Object msgId);
}
```

- Отправка сообщения с msgId

```
_collector.emit(new Values("field1", "field2", 3) , msgId);
```

Болты с сохранением состояния

- Часто нам требуется хранить состояние узла топологии между вызовами.
- Расширяем класс BaseStatefulBolt
- Реализуем дополнительный метод initState в котором сохраняем ссылку на состояние болта.
- В процессе обработки tuple изменяем состояние
- “За кадром” сам storm посылает технологический tuple который обрабатывается болтом который сохраняет текущее состояние в хранилище и выдает подтверждение.

Anchoring

- Anchoring – привязка сгенерированного tuple к оригинальному tuple поступившему на вход
- С случае если требуется узнать о проблеме в обработке любой ветви графа – используется anchoring

```
public void execute(Tuple tuple) {
    ...
     _collector.emit(tuple, new Values(word));
   ….
     _collector.ack(tuple);
}
```

Пример создания локальной топологии

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("generator", new SimplePollSpout());
builder.setBolt("splitter", new SimpleSplitBolt(), 10).shuffleGrouping("generator", "words");
builder.setBolt("counter", new WordCountBolt(), 10)
        .fieldsGrouping("splitter", new Fields("word"))
        .allGrouping("generator", "sync");
Config conf = new Config();
conf.put(SimplePollSpout.POLL_DIR, args[0]);
conf.put(SimplePollSpout.PROCESSED_DIR, args[1]);
conf.setDebug(false);
conf.put(Config.TOPOLOGY_MAX_SPOUT_PENDING, 5);
LocalCluster cluster = new LocalCluster();
cluster.submitTopology("test topology", conf, builder.createTopology());
```

Графическое представление примера