Introduzindo o Apache Storm
MAC5742 - Computação Paralela e Distribuída
Instituto de Matemática e Estatística
Universidade de São Paulo

Lucas Vasconcelos Santana
Prof. Dr. Alfredo Alfredo Goldman vel Lejbman
Ms. Eng. Marcos Amarís González

28 de Junho de 2015

Conteúdo

Lista de Figuras

1 Introdução

O Storm é uma ferramenta distrubuída para computação em tempo-real. Ele pode ser utilizado em áreas como realtime analytics, machine learning e computação contínua, conseguindo consumir e processar grande quantidade de dados e também oferecer confiabilidade nesse processamento.

Tem como pontos importantes a tolerância a falhas e a escalabilidade semelhante ao Apache Hadoop, e também a possibilidade de se utilizar qualquer linguagem no desenvolvimento do seu código Storm.

Atualmente é um projeto de código aberto e também está no rol de projetos TopLevel da fundação Apache. Possui a maior parte do seu código escrito em Clojure e é executado em uma JVM.

Um cluster Storm executa uma topologia, que possui a estrutura de um grafo acíclico dirigido. A topologia engloba a entrada dos dados e todos os processamentos executados nela. Pode haver mais de uma topologia sendo executada em um cluster.

2 Cluster Storm

Em um cluster Storm existem dois tipos de nós: master e workers. O nó master, chamado de **Nimbus**, é responsável pela distribuição das topologias e tarefas no cluster e também monitorar por falhas nos workers, que são responsáveis por executar o processamento necessário.

Cada nó do tipo worker possui um *daemon* chamado **Supervisor**, que é responsável por atender as demandas do nó master, sendo capaz de criar e destruir *workers* quando necessário.

A coordenação do cluster e a correta comunicação entre os nós é feita com auxílio da ferramenta **Zookeeper**. O Zookeeper entrega a funcionalidade de centralização de configurações, fazendo com que seja possível coordenar o cluster do Storm. Com isso, os componentes do Storm podem ser *Stateless*, utilizando o Zookeeper para manter seus estados, aumentando a tolerância a falhas. A estrutura de um cluster Storm pode ser verificada na Figura 1.

3 Topologias

Para processar dados usando o Storm é necessário criar uma **topologia**. Como foi citado anteriormente, uma topologia é um grafo acíclico dirigido, onde nos vértices ocorrem os processamentos dos dados e as arestas indicam transferência de dados entre os vértices.

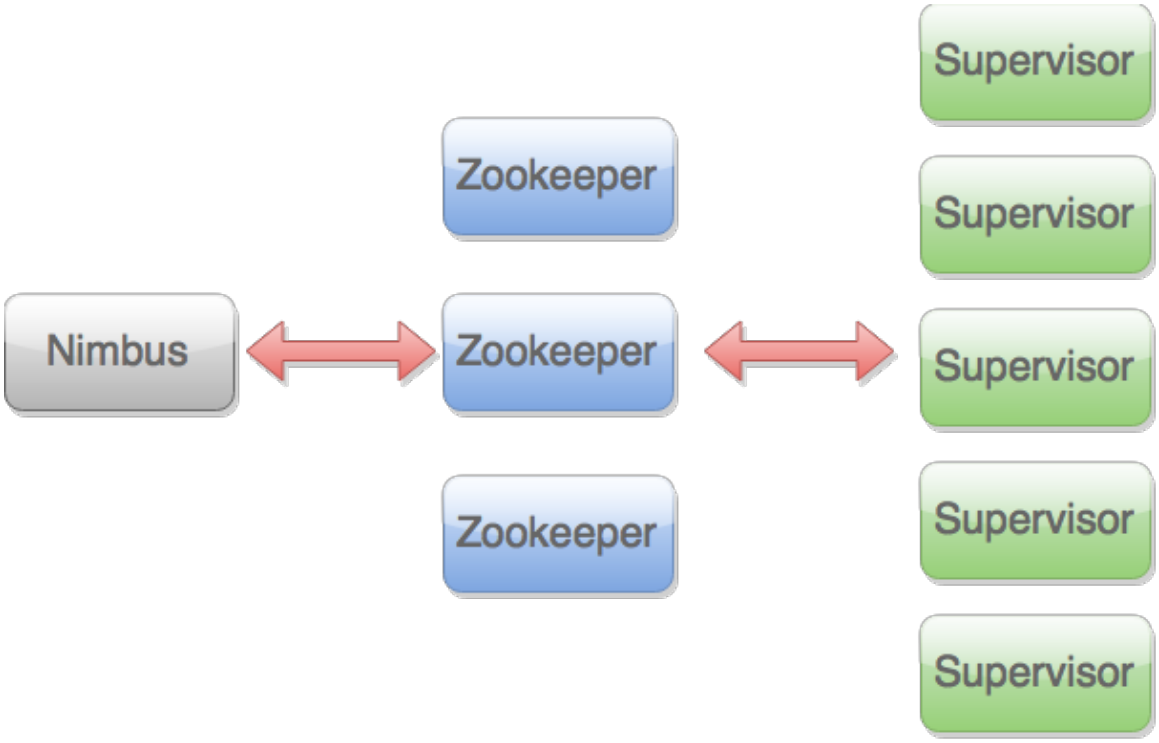

Figura 1: Cluster Storm

O modelo de dados utilizado pelo Storm para ser trafegado nas arestas do grafo são *tuplas*.

Na topologia do Storm, podemos encontrar dois tipos de vértices. Um deles é chamado de **Spout** e o outro é chamado de **Bolt**.

3.1 Spouts

Os Spouts são responsáveis pela geração das tuplas que serão processadas. Esse vértice deve coletar dados de alguma fonte e repassar para os próximos vértices do grafo.

Como um Spout é feito sistemicamente pelo desenvolvedor, os dados podem ter origem de qualquer fonte, seja ela uma fila, uma API ou um log de uma aplicação.

3.2 Bolts

Nos Bolts acontecem o processamentos dos dados de fato. Eles podem executar filtragens, transformações e/ou funções nos dados, se conectar com banco de dados e efetuar transações.

Um bolt deve ser o mais específico possível, como o método de uma classe, seguindo o mesmo conceito de Programação Orientada a Objetos. Assim, podemos ter mais de um bolt em uma topologia Storm, onde um bolt que recebe os dados de um spout pode, por sua vez, passar um novo stream de dados já processados para um outro bolt da topologia executar novos processamentos.

A visualização de um exemplo de topologia Storm é visto na Figura 2. Os **Spouts** são os vértices representados com o ícone de uma *torneira* e os **Bolts** são representados pela figura de um *raio*. E, como dito anteriormente, as arestas representam os dados (tuplas) sendo transferidas de um nó a outro.

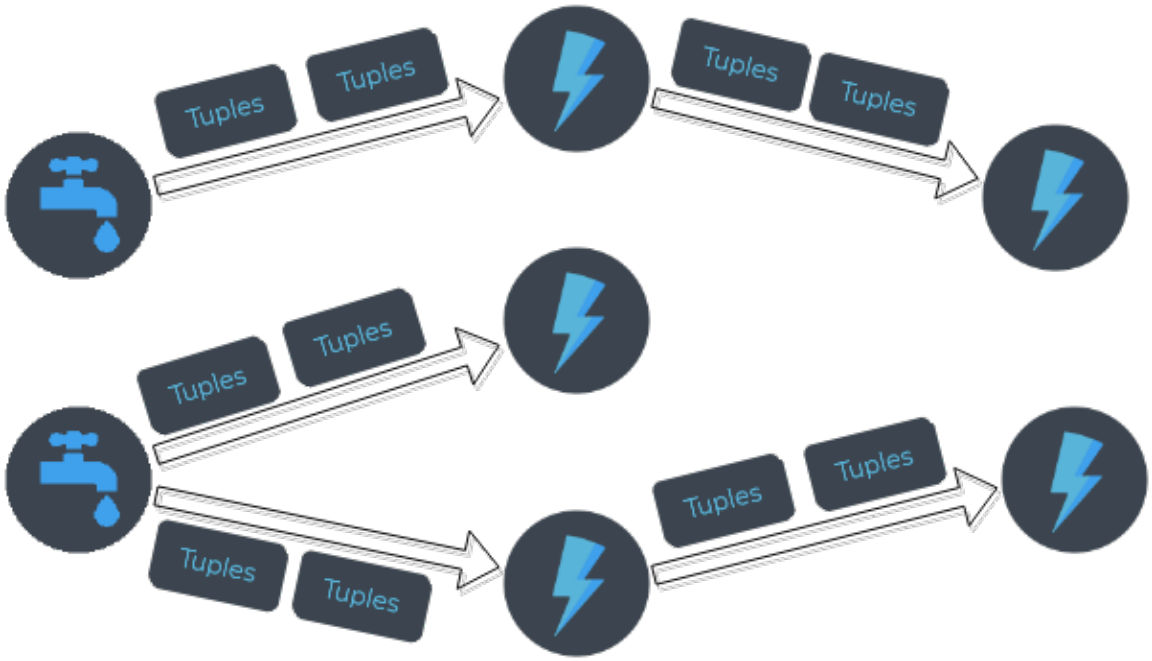

Figura 2: Topologia Storm

3.3 Tasks e Workers

Quando uma topologia é executada, todos os spouts e bolts dela executam as chamadas **Tasks**. Cada task é equivalente à uma thread Java e executa uma instância do spout ou bolt. A quantidade de tasks executada é definida na topologia e pode ser alterada em *runtime*.

Workers são processos executados pelos Supervisors através do Cluster Storm. São eles que são responsáveis pela execução das tasks dos spouts e bolts. Em um nó

do tipo Supervisor, pode haver um ou mais workers. Essa quantidade é informada na definicação da topologia e também pode ser modificada em *runtime*.

3.4 Groupings

Ao desenvolver um bolt, deve-se especificar quem será a fonte de recebimento dos dados, seja ela um spout ou até mesmo outro bolt. No Storm, isso é feito através dos **Groupings**, que dizem que comportamento o envio de dados entre os vértices da topologia terá.

Por padrão, o Storm define sete tipos de Groupings para transferência dos dados. São eles:

- **Suffle grouping**: Cada bolt gerado no cluster (definido pelo número de paralelismo utilizado) receberá a mesma quantidade de tuplas de maneira aleatória;
- **Fields grouping**: Através de funções geradoras as tuplas são distribuídas para uma determinado bolt. Uma função pode ser um mod hashing, por exemplo;
- **Partial Key grouping**: Semelhante ao *Fields grouping*, porém oferece melhor balanceamento do stream de dados;
- **All grouping**: Todos as tasks dos bolts receberão as mesmas tuplas, havendo replicação de dados entre todas elas;
- **Global grouping**: Apenas uma task de um bolt receberá todos os dados;
- **None grouping**: Semelhante ao *Shuffle grouping*;
- **Direct grouping**: Quando quem envia a tupla (spout ou bolt) define exatamente quem deve receber os dados;

4 Garantindo o processamento de uma tupla

O Apache Storm garante que uma tupla que se originou de um spout será totalmente processada até o fim da topologia. Uma tupla é considerada processada quando chega ao final da topologia e nenhuma tupla é gerada a partir dela.

Quando uma tupla se origina em um Spout, a cada vez que ela é processada uma ou mais tuplas são geradas a partir dela. Assim, o Storm consegue construir a árvore de tuplas que uma única tupla originou até o momento que ela é totalmente processada. Dessa maneira, é possível para um Spout verificar se uma determinada tupla foi processada e, se houve sucesso, confirmar com um *ack*.

5 Paralelismo em uma topologia

Na execução de uma topologia, podemos distinguir os *Workers*, que geram as *Threads (também chamadas de Executors)* que, por sua vez, executam as *Tasks*. Uma thread pode executar uma ou mais tasks.

Um worker pode executar toda ou parte de uma topologia, dependendo de como ela foi definida e da arquitetura do cluster onde ele se encontra.

Ao definir um spout ou um bolt, é possível informar ao Storm a quantidade de paralelismo que deverá ser aplicado, ou seja, a quantidade de threads que terão uma task daquele spout ou bolt.

A Figura 3 mostra uma topologia com 1 spout e 2 bolts sendo executada. Nessa topologia, o spout *Blue* e o bolt *Green* estão configurados para ter um nível de paralelismo 2, e o bolt *Yellow* possui o paralelismo de 6.

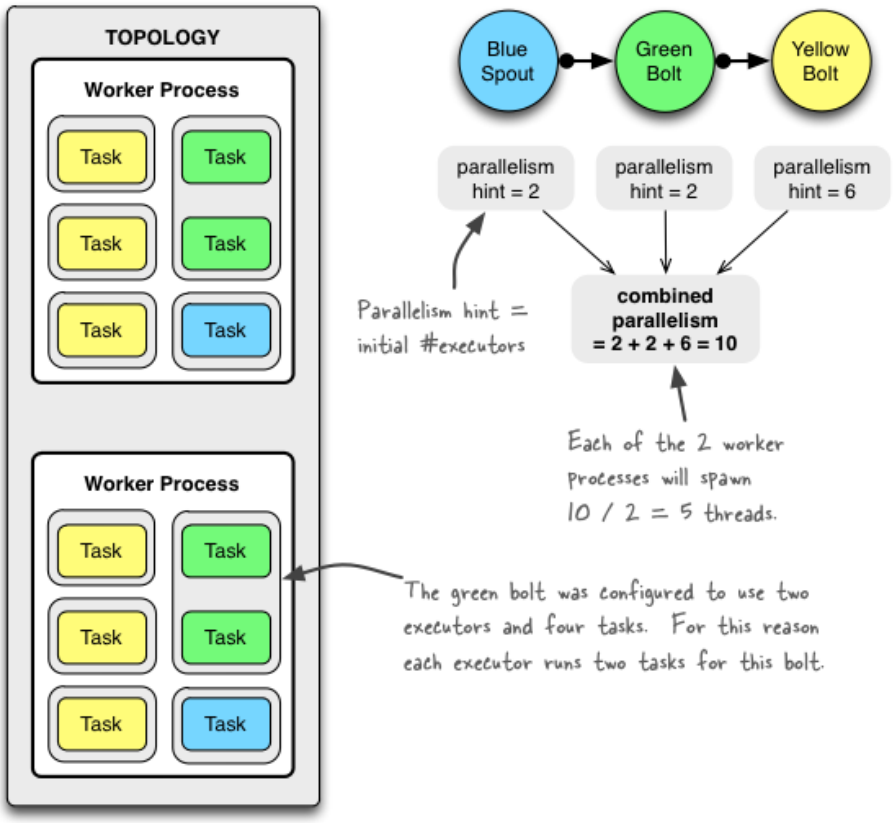

Figura 3: Topologia Storm sendo executada. Retirada de: http://storm.apache.org/documentation/Understanding-the-parallelism-of-a-Storm-topology.html

6 Desenvolvimento de uma topologia

Para este exemplo de execução de uma topologia, apenas a título de demonstração, será utilizada uma topologia simples, constituída de 1 spout e 2 bolts. Onde o Spout será responsável por enviar *strings* aleatórias para um bolt que irá adicionar três pontos de exclamação ao final da *string* e passar a saída para uma outra instância desse mesmo bolt, que irá adicionar mais três pontos de exclamação à *string* recebida. Assim, ao final do processamento, a tupla inicial contendo uma palavra, irá finalizar com seis pontos de exclamação adicionados.

6.1 Desenvolvendo o Spout

No desenvolvimento do Spout, devemos utilizar a interface nativa fornecida pelo Storm, chamada **IRichSpout**. Nesse exemplo, o spout *TestWordSpout* estende a classe *BaseRichSpout*. Ele envia, aleatoriamente, uma *string* para o próximo passo da topologia, que será consumida por um bolt. O código a seguir mostra a implementação desse spout. Alguns detalhes não importantes para o exemplo foram omitidos.

```
public class TestWordSpout extends BaseRichSpout {
    SpoutOutputCollector _collector;

    ...

    // Metodo nextTuple() chamado pela interface IRichSpout
    // para cada tupla lida. No final do metodo e enviada
    // a tupla com o metodo emit
    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));
    }

    // Metodo utilizado para verficiar se uma determinada
    // tupla chegou ao final do processamento
    public void ack(Object msgId) {

    }

    ...

}
```

Foram mantidos os métodos *nextTuple()* e *ack()*. Ambos são utilizados pela interface nativa de Spouts do Storm. Nesse caso, o Storm chama o método *nextTuple()* que emitirá uma *string* para a topologia e, o método *ack()* será chamado para verificar se aquela determinada tupla foi totalmente processada, podendo chamar novamente *nextTuple()*.

6.2 Desenvolvendo o Bolt

Da mesma maneira que o Storm faz com os spouts, ele também oferece interfaces e classes nativas para uso com Bolts. Nesse exemplo, o bolt *ExclamationBolt* implementa a interface *IRichBolt*. O bolt mostrado a seguir recebe uma tupla que, no caso, será uma *string* recebida do spout *TestWordSpout*, e irá acrescentar três pontos de exclamação no final dela.

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

    ...

    // Metodo chamado pelo Storm quando o bolt recebe uma tupla
    // Onde e definido o processamento desse dado
    public void execute(Tuple tuple) {
        _collector.emit(tuple, new
                            Values(tuple.getString(0) + "!!!"));
        _collector.ack(tuple);
    }

    ...

}
```

Em um bolt, o método executado pelo storm é o *execute()*, que recebe uma tupla como parametro e faz o processamento dela e emite novamente para a topologia. Se ninguém receber essa tupla gerada, a árvore gerada pela tupla original (vinda do spout) finaliza e ela é considerada totalmente processada.

6.3 Executando a topologia

Com spouts e bolts escritos, pode-se escrever e executar a topologia desenvolvida neste exemplo. Para isso, foi usada a classe oferecida pelo Storm, chamada *TopologyBuilder*. Nessa topologia, é setado o spout definido anteriormente *TestWordSpout* com o *id* 'word', um bolt do tipo *ExclamationBolt* com o *id* 'exclaim1' recebendo os dados do spout e, por fim, um outro bolt do tipo *ExclamationBolt* possuindo *id* 'exclaim2' recebendo os dados gerados pelo bolt 'exclaim1'.

Segue o código utilizado na execução da topologia, onde são utilizados os métodos *setSpout* e *setBolt* da classe *TopologyBuilder* para definir os spouts e bolts da topologia. Um detalhe destes métodos é o último parâmetro passado para eles, que define o nível de **paralelismo** que será utilizado na execução de cada processamento dessa topologia.

```
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");
```

Assim, nessa topologia, o spout 'words' terá paralelismo de valor 10, o bolt 'exclaim1' terá paralelismo de valor 3 e o bolt 'exclaim2' terá paralelismo de 2.

7 Conclusões

O Apache Storm oferece funcionalidades de tolerância a falhas e confiabilidade no processamento dos dados que de fato funcionam. O desenvolvimento em cima dele não é trivial, exigindo um conhecimento no mínimo intermediário do ambiente para a criação de topologias robustas. O uso de outras linguagens no Storm também pode ser complexo e contribui com isso o fato do Storm oferecer suporte nativo apenas para Python, Ruby e Scala, sendo assim necessário o desenvolvimento de adaptadores para outras linguagens.

Um outro ponto positivo é o ambiente de desenvolvimento oferecido por ele, onde um Cluster Storm pode ser simulado, ajudando nos estudos.

Levando em conta sua robustez e performance, o Storm pode ser uma boa alternativa ao **Apache Hadoop** ou ao **Apache Spark**. Na Figura 4 existe uma tabela com algumas comparações entre o Storm e o Spark.

	Apache Storm	*Apache Spark*
1	Task Parallel Computations	Data Parallel Computations
2	Stream processing engine which can undergo micro-batching	Batch processing engine with micro batches but no streaming
3	Fault tolerant method for multiple computations on an event with each event dealt with individually	Use of Resilient Distributed Datasets, which are immutable, for parallel operations
4	Record-at-a-time processing model	Processes through mini batches
5	Latency is sub-second	Latency is few seconds
6	A better option in case of no data loss wanted	A good choice for a computation, in which each event is processed exactly once
7	Implemented in Clojure	Implemented in Scala
8	Comes with a Java API	Can be programmed in Scala as well as Java
9	Storm runs on Mesos	Spark runs on Mesos and YARN
10	Operates on data in motion i.e. continuous streaming of data	Operates on data at rest

Figura 4: Algumas diferenças entre o Apache Storm e o Apache Spark. Retirada de: http://blog.spec-india.com/apache-storm-vs-apache-spark-an-overall-comparison

8 Referências

Referências

[1] Apache foundation announces storm as a toplevel project. `https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces64`. Acessado: 27-06-2015.

[2] Apache storm oficial website. `http://storm.apache.org`. Acessado: 27-06-2015.

[3] Apache storm vs. apache spark. `http://blog.spec-india.com/apache-storm-vs-apache-spark-an-overall-comparison`. Acessado: 27-06-2015.

[4] History of apache storm and lessons learned. `http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html`. Acessado: 27-06-2015.

[5] Storm introduction. http://www.infoq.com/presentations/Storm-Introduction. Acessado: 27-06-2015.