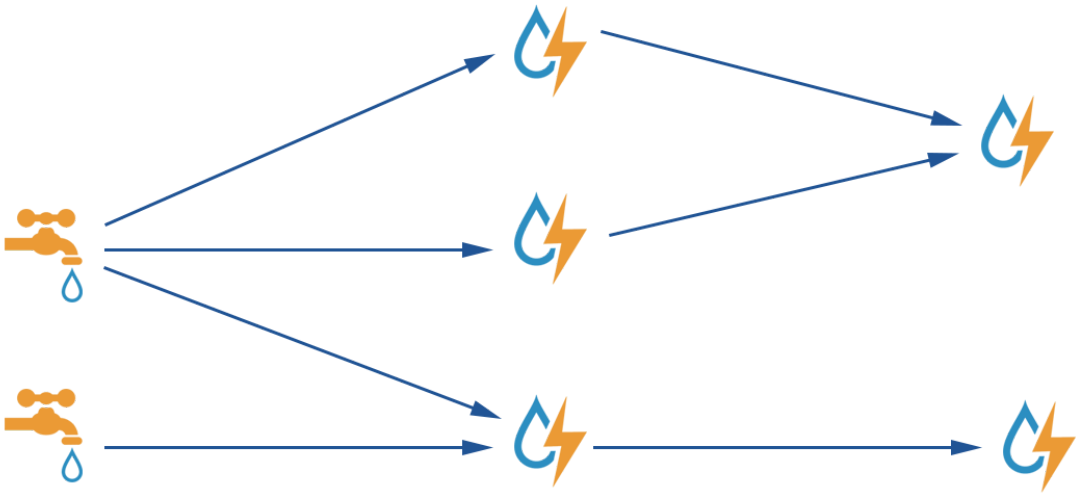
Felipe Izaguirre 205984
Giovane Ribeiro 252865

Processamento de Dados de Estações Climáticas com Apache Storm e Microsoft Azure

Apache Storm Introdução

Apache Storm é um sistema distribuído de processamento de dados em tempo real que implementa aplicações em forma de topologias. Essas topologias são representações de grafos acíclicos dirigidos (DAG) onde cada vértice realiza um processamento distinto e repassa os dados ao próximo vértice ou finaliza o processamento salvando no banco de dados ou realizando alguma outra operação de saída para mostrar ao usuário os resultados.



Uma topologia na plataforma Apache Storm é constituída de Spouts e Bolts. Spouts são as entradas da topologia, e bolts as unidades de processamento. Diferentemente de outros frameworks como Hadoop ou Spark, Storm não possui uma tarefa a ser executada com início e fim, uma topologia que é criada vai estar disponível a qualquer momento e só será fechada caso o usuário termine sua execução.

Um cluster Storm é composto por 3 componentes, sendo eles:

Nimbus: Servidor mestre, responsável por distribuir as tarefas.

Zookeeper: Responsável por guardar o estado do cluster.

Supervisor: Possui um ou mais processos, e é responsável por delegar as tarefas aos processos.

Escopo do Projeto

Projetos acadêmicos já em andamento tem como objetivo o desenvolvimento de estações de monitoramento climático de baixo custo que possa ser monitoradas remotamente. A falta de ferramentas eficientes para análise dos dados gerados por estas estações motivou o desenvolvimento de uma aplicação streaming que fosse capaz de consumir os dados e processá-los em tempo real, também com capacidade de escalabilidade em cluster. Cada estação meteorológica deverá ser capaz de monitorar e enviar ao cluster: (i) A temperatura ambiente (celsius); (ii) A pressão atmosférica (hPa); (iii) Luminosidade (Lumens); (iv) Umidade Relativa do ar. Através do processamento desses dados pretende-se efetuar a análise desses dados a fim de adquirir informações a respeito do clima.

Dessa forma, a partir das informações recebidas das estações climáticas é calculado em tempo real o Índice de Calor, Condição do Tempo(Nublado, Ensolarado, Noite), Condição de pressão atmosférica e o Índice de Humidade Relativa.

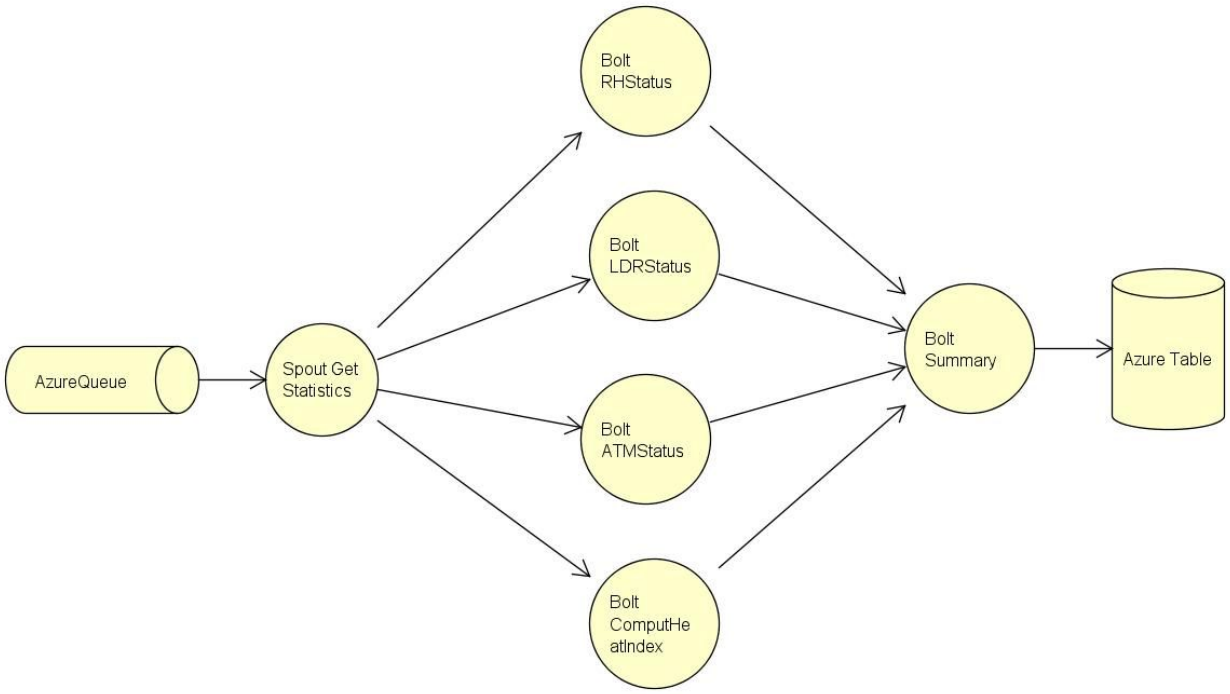
Devido a falta de estações disponíveis para consulta dos dados, foram assumidas algumas simplificações quanto a comunicação entre as estações e o cluster. Com um foco na aplicação Storm esses dados foram simulados e enviados para a aplicação utilizando o seguinte formato de json baseado no trabalho anterior de Leandro Bruscato.

{"sensors": {"DHT22_AH": 63.67657145663951, "LDR": 34.810992385681054, "BMP085_PRESSURE": 97678.4227262929, "DHT22_TEMP": 28.760296662135584}, "datetime": {"source": "RTC_DS1307", "stationID": 1, "value": "2017-07-29-14-07-35", "format": "%Y-%m-%d-%H-%M-%S"}}

Foi adicionado o campo *stationID* para que fosse possível identificar a origem do dado recebido.

Topologia da Solução

A figura a seguir ilustra a topologia implementada utilizando o Apache Storm. Como podemos ver existem cinco bolts e um spout.



Topologia Storm para Processar dados Climáticos

Spout: Como não existem estações climáticas em funcionamento disponíveis, foi gerado dados de modo aleatório no mesmo formato JSON a ser disponibilizado pelas estações, com o aditivo do campo stadionID que não existe no original mas que para a nossa implementação se fazia necessário. Esses são dados colocados em uma fila disponível como serviço do Microsoft Azure. Desse modo o spout implementado lê o itens da fila e os distribui para os bolts.

Bolt RHStatus: Cada bolt é responsável por um processamento diferente, o bolt *RHStatus* processa os dados com base nos índices de umidade aplicando níveis de alerta OK, Attention, Alert e Emergency. Após o seu processamento o bolt *RHStatus* publica a tupla processada para que o próximo bolt efetue o seu processamento. Os outros bolts atuam de maneira semelhante, mudando apenas o tipo de processamento realizado.

Bolt LDRStatus: Este bolt retorna como resultado de suas operações Sunny, Mood e Night de acordo com o nível de luz no ambiente.

Bolt ATMLevel: Através da pressão atmosférica, este bolt retorna valores High ou Low, assim publicando se a pressão é alta ou baixa.

ComputeHeatIndex: O índice de calor resulta de um cálculo relacionando a temperatura e humidade, e este bolt é responsável por esta avaliação. Além disso, baseado no índice de calor informa se há algum risco a saúde através dos seguintes status: OK, CAUTION, EXTREMELY_CAUTION, DANGER e EXTREMELY_DANGER.

Bolt SummaryResults: Como se pode perceber pelas ilustrações, um bolt pode ter mais de uma fonte de dados. Assim foi criado um bolt para analisar a saída de cada um dos bolts anteriores, desta forma o SummaryResults é responsável por analisar todos os resultados de cada um dos bolts, agregá-los e salvá-los em uma tabela no banco de dados NoSQL da Azure.

No momento da inicialização é passado um número de tarefas para cada Spout/Bolt. Essas tarefas são executadas em executores e os executores são criados por processos no cluster. As tarefas são executadas em série nos executores, sendo que o # executores <= # tarefas. Isto dá flexibilidade para rebalancear a topologia com o cluster em execução, sem necessidade de parada. Por padrão a nossa topologia configura o seguinte número de tarefas:

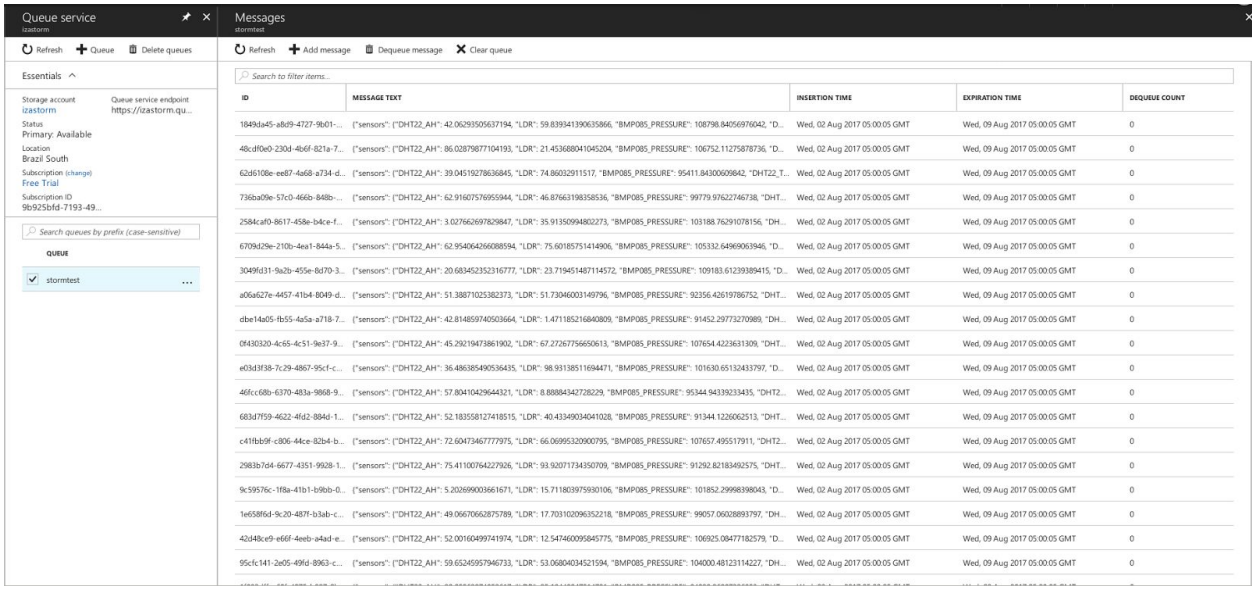
get-statistics Spout = 2 tarefas
atm-status Bolt = compute-heat-index = ldr-status = rh-status = 4 tarefas
summary Bolt = 8 tarefas

Recursos da Nuvem

Usamos recursos da nuvem para este trabalho a fim de explorar ao máximo os recursos de Big Data em um cluster com recursos distribuídos em um ambiente potente. Para auxiliar na entrada e saída dos dados, utilizamos o recurso de Fila disponibilizado pela Azure chamado Azure Queue, e para a saída dos dados utilizamos uma banco de dados No-SQL disponibilizado pela Azure chamado Azure Table. Abaixo relatamos a função destes dois elementos de forma mais detalhada.

Azure Queue: Scripts em python simulam estações climáticas, as informações geradas pelas estações são enviadas para uma fila na nuvem através de chamadas utilizando a API disponibilizada pela Azure.

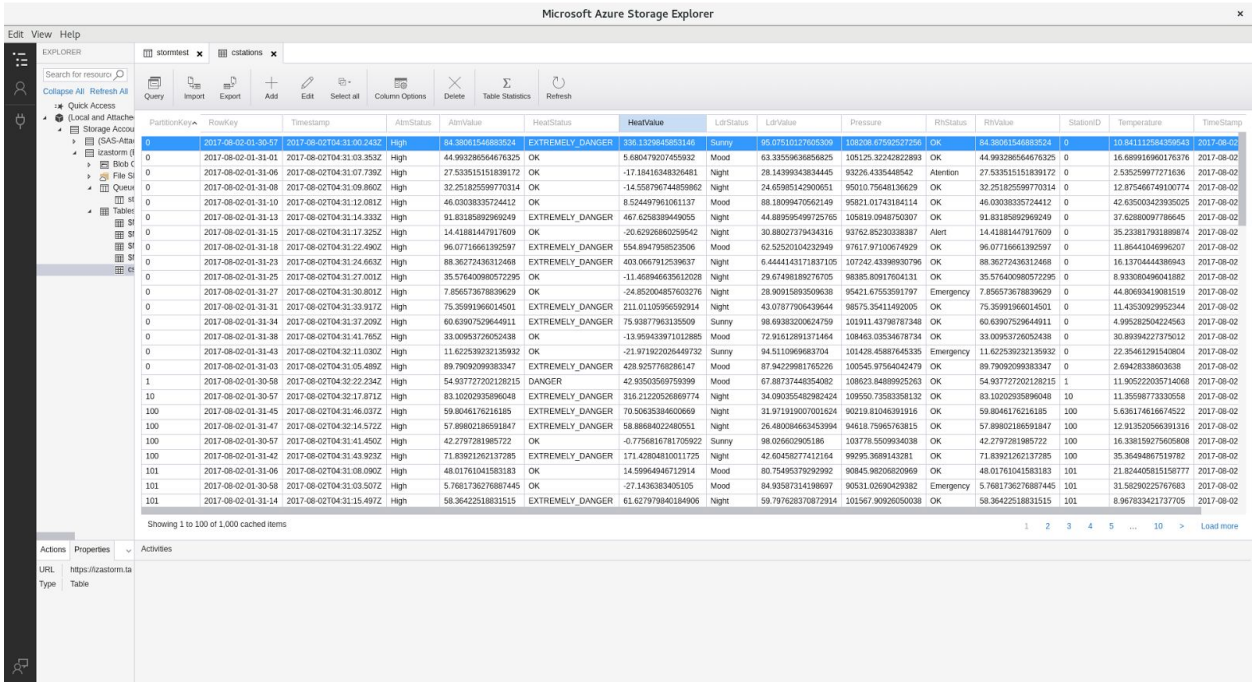
A figura abaixo exibe as mensagens que estão na fila.



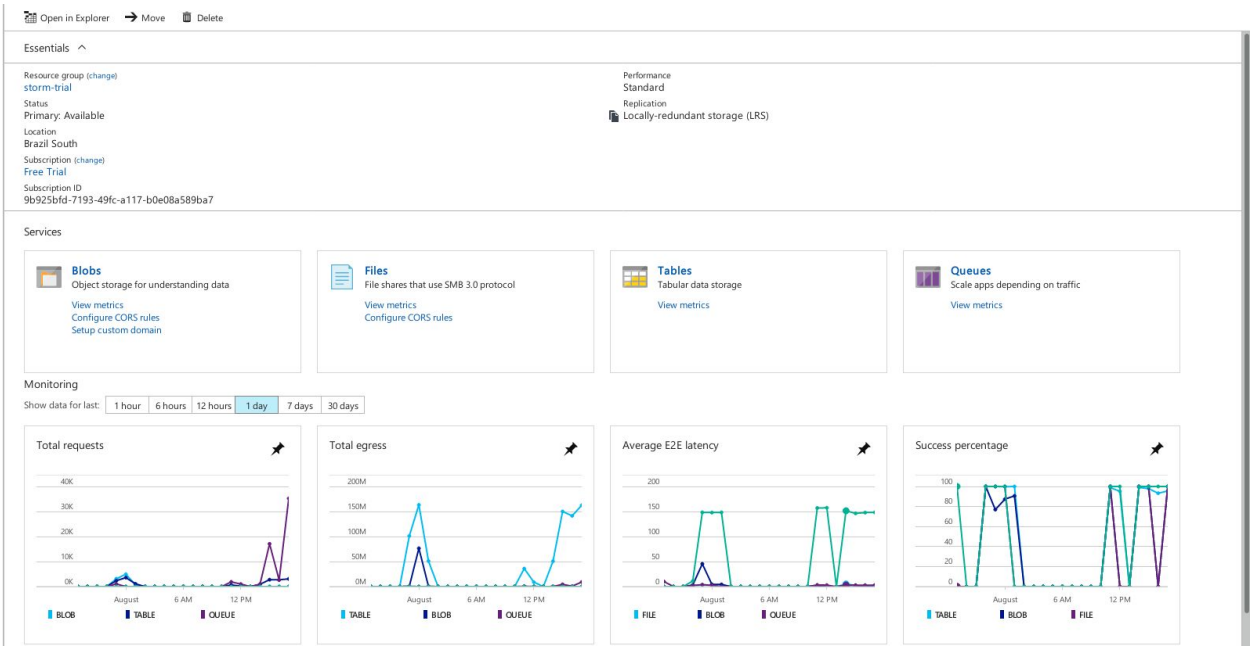
Azure Table: O bolt *SummaryResults* após receber a saída de um processamento completo de uma informação da estação, envia os dados para uma tabela em um banco de dados No-SQL. Esta tabela contém os seguintes campos: Partition Key, Row Key, TimeStamp, AtmStatus, HeatStatus, HeatValue, LdrStatus, LdrValue, Pressure, RhStatus, RhValue, StationID, Temperature, TimeStamp.

Os dois primeiros campos são os mais importantes, esses dois campos são utilizados para identificar uma única linha na tabela. Sendo assim é recomendado que a Partition Key tenha elementos duplicados, enquanto que Row Key elementos únicos para uma determinada Partition Key. Desta forma o banco de dados otimiza as consultas e inserções na tabela. O terceiro campo Timestamp é preenchido de forma automática pelo banco de dados, e informa o horário em que a entrada foi adicionada. Os outros campos são referentes às saídas dos bolts. O último campo que se difere no nome do terceiro apenas pelo S maiúsculo, é o horário que foi enviado da estação climática, ou seja, o momento em que ela enviou as informações para a fila de entrada.

Abaixo podemos ver uma imagem da tabela descrita nesta seção.

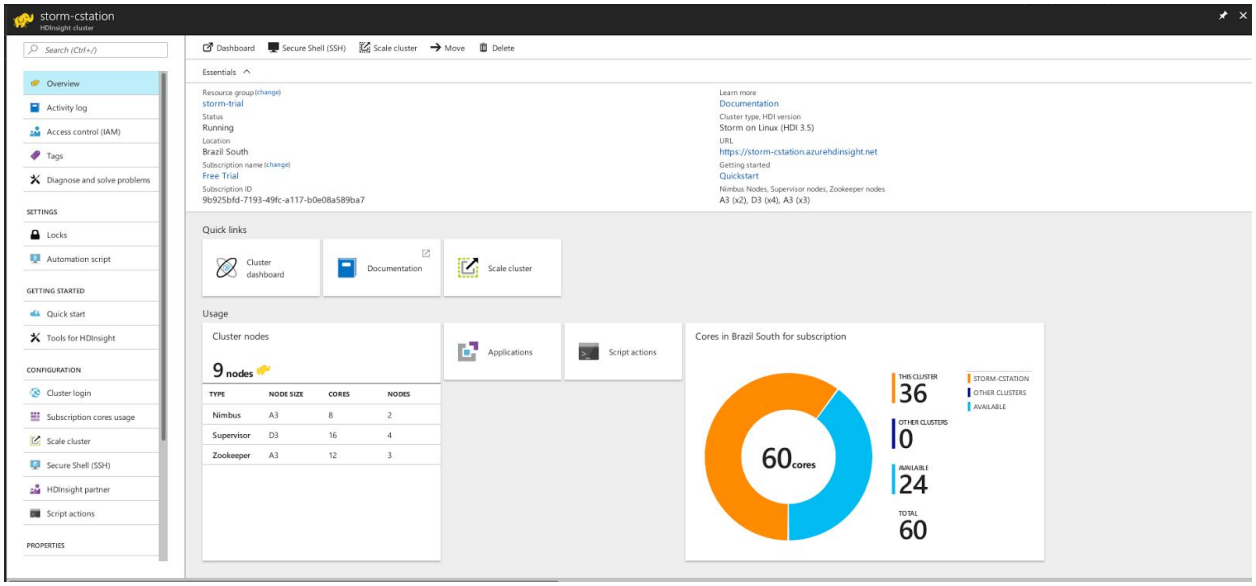


Abaixo é possível conferir o screenshot da Dashboard com os serviços de Fila e Tabela mencionados acima.



Por fim o principal serviço utilizado foi o *HDInsight*, que é o serviço que disponibiliza um cluster Storm, para podermos executar a topologia. Ao criar o cluster é perguntado o número de nodos, e o tipo de recurso que será utilizado em cada nodo

em termos de CPU, Memória e Disco. Abaixo está um screenshot de como é a tela de administração do serviço.



Ambiente de Desenvolvimento

A implementação da topologia foi feita inteiramente utilizando Java. Para gerenciar as dependências e e geração do jar foi utilizado o utilitário Maven.
Para compilar o código é necessário ter o Java SDK 8.x instalado assim como o Maven 3.x e então na pasta do projeto entre no diretório *collector* e execute:

 # mvn compile

 # mvn assembly:assembly

No diretório target será gerado dois jars, o que diz collector-0.0.1-SNAPSHOT-jar-with-dependencies.jar é o pacote que deve ser enviado para ser executado no cluster.
Para executar o cluster execute:

 # storm jar collector-0.0.1-SNAPSHOT-j-with-dependencies.jar csc.collector.TopologyMain

Pode ser passado como parâmetro adicional a palavra *local* para executar o cluster localmente.

O simulador das estações foi feito em Python 2.7, para executá-lo é necessário o pacote azure.storage, para instalá-lo utilizando o pip execute a linha abaixo no terminal:

 # pip install azure

Com o pacote da azure script pode ser executado passando como argumento o intervalo em segundos que as informações serão enviadas, o número de estações e um inteiro que representa um deslocamento no id das estações. Sendo assim ao executarmos:

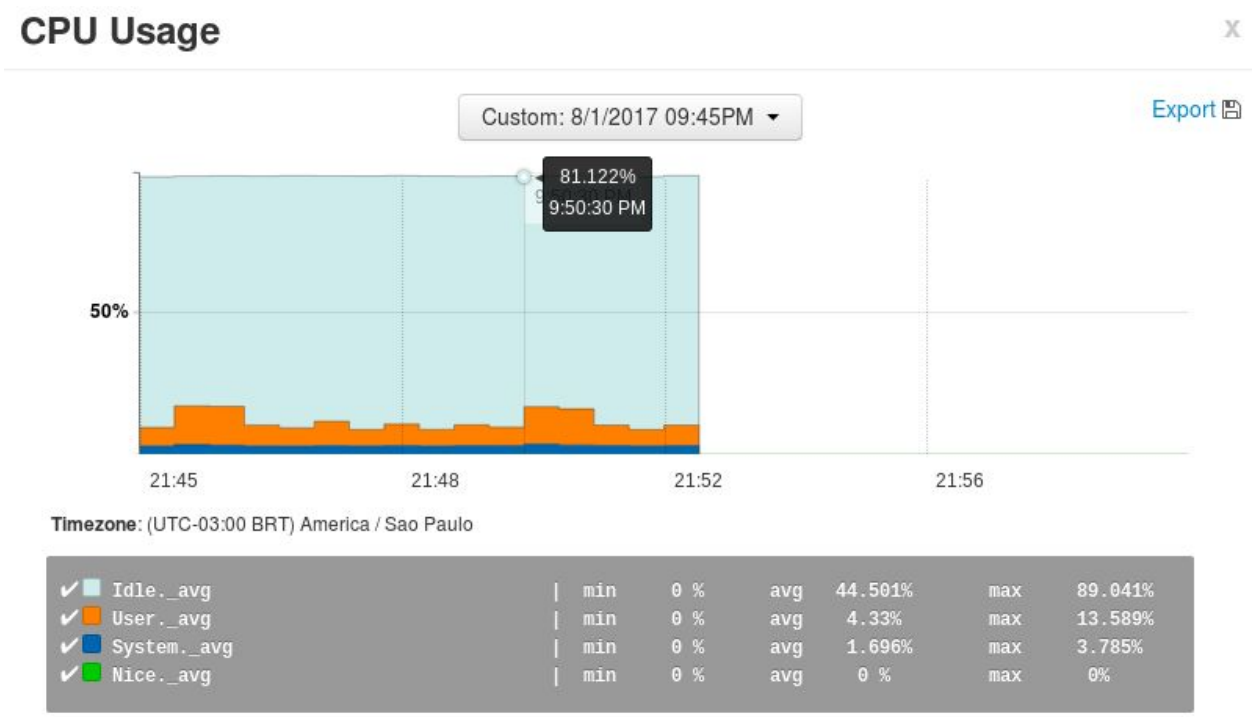
 # ./jsonGenerator.py 30 1000 0

Estamos informando para que utilize 30 segundos de intervalo no envio das informações, sejam simuladas 1000 estações e que o id comece a partir de 0.

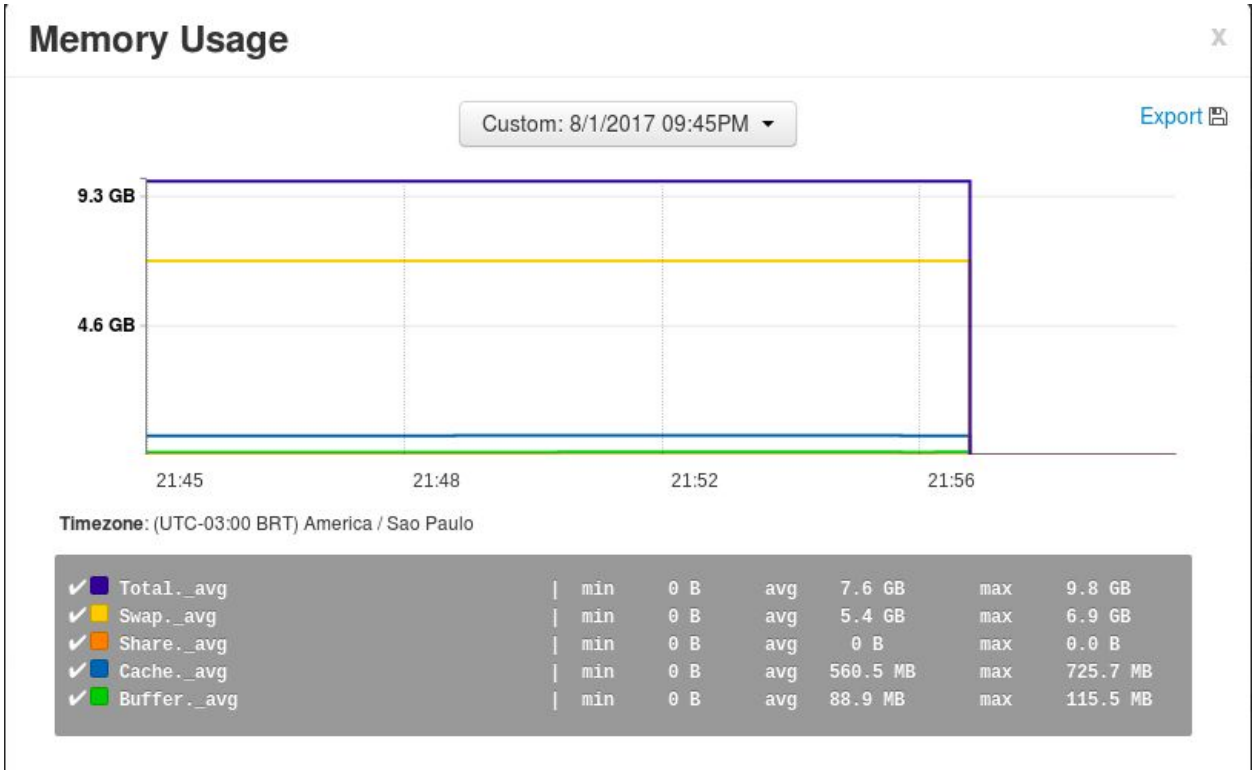
Experimentos

Em um cluster composto por Nimbus (2 x A3), Supervisor (4 x D3), Zookeeper (3 x A3). Simulamos 100 estações enviado dados de 30 em 30 segundos, e obtivemos os seguintes dados.

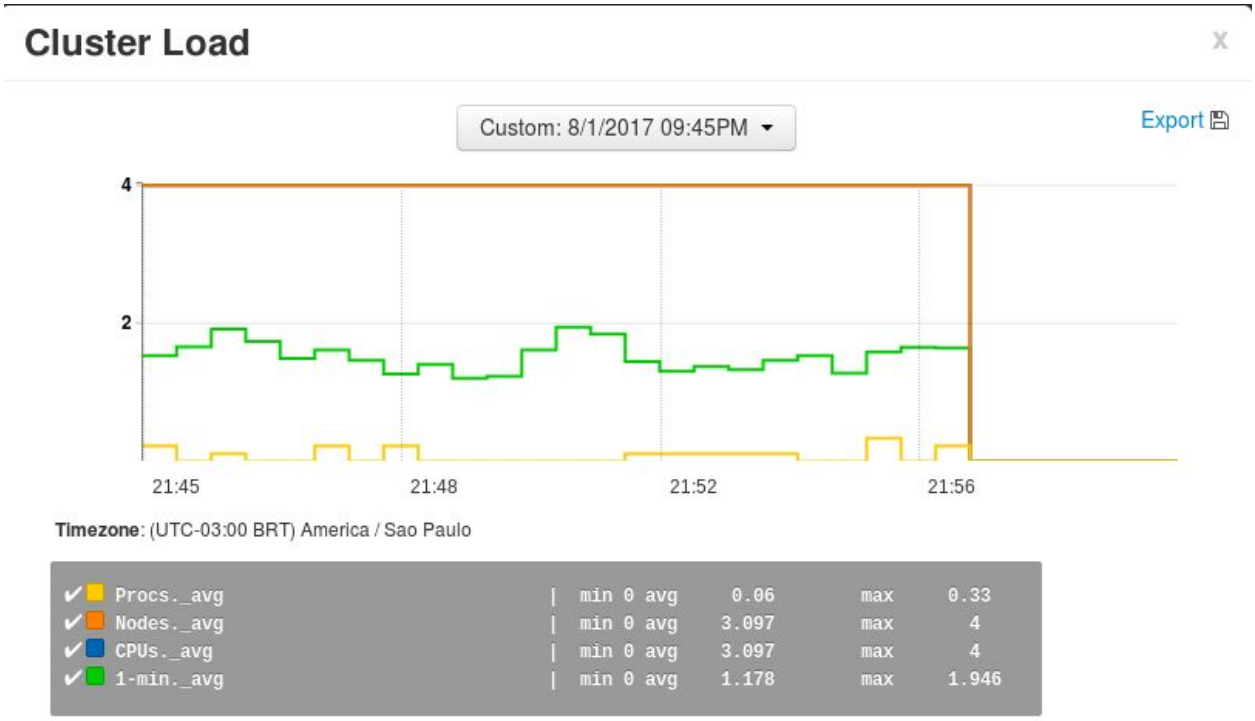
CPU: O uso da CPU ficou em torno de 6% de uso.



Memória: Uso da memória se manteve estável com 9,4 GB sendo usado.



Cluster Load: A carga no cluster ficou em torno de 1,54.



As ilustrações acima são apenas para demonstrar como medimos o desempenho, para os próximos cenários utilizaremos a tabela abaixo.

Os experimentos foram feitos levando em conta as configuraçõe abaixo:

Caso 1: Nimbus (2 x A3), Supervisor (4 x D3), Zookeeper (3 x A3)
get-statistic = 2 executores
atm-status = compute-heat-index = ldr-status = rh-status = 4 executores
summary = 8 executores

Caso 2: Nimbus (2 x A3), Supervisor (4 x D3), Zookeeper (3 x A3)
get-statistic Spout = 4 executores
atm-status = compute-heat-index = ldr-status = rh-status = 8 executores
summary = 16 executores

No momento da inicialização é passado um número de tarefas para cada Spout/Bolt. Essas tarefas são executadas em executores e os executores são criados por processos no cluster. As tarefas são executadas em série nos executores, sendo que o # executores <= # tarefas. Isto dá flexibilidade para rebalancear a topologia com

o cluster em execução, sem necessidade de parada. Por padrão a nossa topologia configura o seguinte número de tarefas:

get-statistics Spout = 2 tarefas

atm-status Bolt = compute-heat-index = ldr-status = rh-status = 4 tarefas

summary Bolt = 8 tarefas

Configuração do Cluster	Número de Estações Enviando Informações	CPU	Memória	Cluster Load
Caso 1 Nimbus (2 x A3), Supervisor (4 x D3), Zookeeper (3 x A3)	100	4.33%	7,6GB	1.18
	1000	8.14%	9.8GB	2.74
	2500	7,56%	9.8GB	2,91
	5000	8,45%	9.8GB	3,00
Caso 2 Nimbus (2 x A3), Supervisor (4 x D3), Zookeeper (3 x A3)	100	8,4%	9.8GB	1,42
	1000	7,6%	9.8GB	1,37
	2500	8,04%	9.8GB	1,59
	5000	7,04%	9.8GB	1,77

Tipos de nodo

A3 - 8 cores

D3 - 16 cores

O que podemos perceber destes resultados é que no caso 1, com menor número de executores, a medida que aumentamos a quantidade de estações enviando dados, maior foi a carga no cluster. Em contrapartida no caso 2 ao aumentar o número de estações a carga não se alterou muito. Isto se deve provavelmente ao fato de que o caso 2 explora melhor o paralelismo do cluster, enquanto que no caso 1, utilizando um número menor de executores, o cluster precisa trabalhar mais para dar conta de processar todas as entradas que estão chegando, aumentando assim a carga no cluster.

Não realizamos experimentos com um número maior de estações devido ao fato de que o serviço de fila começou a retornar time out com 5000 estações, isto pode ser

uma limitação do serviço, para um ambiente real seria recomendável o uso de um mecanismo mais robusto para fila, como o Kafka.

Conclusão

O trabalho foi bem bem interessante e desafiador, apresentando utilidade prática em problemas do mundo real.
Durante os desenvolvimento e testes tivemos alguns problemas com a interface da fila e da tabela, em alguns momentos para os testes com mais de 2500 estações eram retornados erros de tentativas excedidas, provavelmente devido ao número de threads que faziam a chamada a API da nuvem ao mesmo tempo.

Entretanto conseguimos implementar uma topologia usando o Storm que pode ser facilmente adaptada para outros tipos de entrada e saída de dados.

Foram enfrentadas dificuldades em relação a curva de aprendizagem do framework e das ferramentas necessárias para o desenvolvimento e utilização do Storm, porém após superadas essas dificuldades, é possível perceber a flexibilidade do framework e sua capacidade para resolver problemas de Big Data. Também, há uma grande variedade de opções de integração para o framework, possibilitando assim a migração e substituição de componentes auxiliares, o que aumenta a flexibilidade e poder de adaptação da ferramenta a diferentes tipos de problemas.

Referencias

http://storm.apache.org/releases/current/index.html
https://docs.microsoft.com/en-us/azure/
http://www.tutorialspoint.com/apache_storm/