MS4S21 - Big Data Engineering and Applications

Supplementary Resources

Moizzah Asif, J418, moizzah.asift@southwales.ac.uk

University of South Wales

Contents

Week 1

1.1 Sharding

Sharding is a method for distributing data across multiple machines.

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives.

There are two methods for addressing system growth: vertical and horizontal scaling.

1.1.1 Vertical Scaling

Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space. Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload. Additionally, Cloud-based providers have hard ceilings based on available hardware configurations. As a result, there is a practical maximum for vertical scaling.

1.1.2 Horizontal Scaling

Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The trade off is increased complexity in infrastructure and maintenance for the deployment.

1.1.3 Shard Keys

The shard key consists of a field or fields that exist in every document in the target collection.

You choose the shard key when sharding a collection. The choice of shard key cannot be changed after sharding. A sharded collection can have only one shard key.To shard a non-empty collection, the collection must have an index that starts with the shard key.

Week 6

Please note most of the content this week has been adapted from AWS resources.

You may want to revisit the lecture videos or handouts from week 4 to revisit the basics of AWS cloud and where does EMR sit with in the big data services.

6.1 Amazon EMR

6.1.1 Introduction and overview

Amazon EMR is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop on AWS to process and analyze vast amounts of data. The main processing frameworks available for Amazon EMR are Hadoop MapReduce and Spark.

By using such frameworks and related open-source projects, like Apache Hive, you can process data for analytics purposes and business intelligence workloads.

Additionally, you can use Amazon EMR to transform and move large amounts of data into and out of other AWS data stores and databases, such as Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB.

Deja vu - Overview

The central component of Amazon EMR is the cluster. A cluster is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances.

Each instance in the cluster is called a node. Each node has a role within the cluster, referred to as the node type.

Amazon EMR also installs different software components on each node type, giving each node a role in a distributed application like **Apache Hadoop**. (The text in bold and onward should ring a bell/s and if doesn't, please revise concepts from weeks 1 - 3).

The node types in Amazon EMR as follows:

- **Master node**:

  A node that manages the cluster by running software components to coordinate the distribution of data and tasks among other nodes for processing. The master node tracks the status of tasks and monitors the health of the cluster. Every cluster has a master node, and it's possible to create a single-node cluster with only the master node.

- **Core node**:

  A node with software components that run tasks and store data in the Hadoop Distributed File System (HDFS) on your cluster. Multi-node clusters have at least one core node.

- **Task node**:

  A node with software components that only runs tasks and does not store data in HDFS. Task nodes are optional.

6.1.2 Understanding the cluster life cycle

A successful Amazon EMR cluster follows the lifecycle specified in the steps below and the following flow chart shown in Fig **??**.

1. Amazon EMR first provisions EC2 instances in the cluster for each instance according to your specifications.For all instances, Amazon EMR uses the default AMI for Amazon EMR or a custom Amazon Linux AMI that you specify.

   During this phase, the cluster state is ***STARTING***.

2. Amazon EMR runs bootstrap actions that you specify on each instance. You can use bootstrap actions to install custom applications and perform customisation that you require. For more information on bootstrapping in an EMR cluster you may find this resource useful. However, please bear in mind that this section is created to give you a brief overview before starting any practical work.

   During this phase, the cluster state is ***BOOTSTRAPPING***.

3. Amazon EMR installs the native applications that you specify when you create the cluster, such as Hive, Hadoop, Spark, and so on.

4. After bootstrap actions are successfully completed and native applications are installed, the cluster state is ***RUNNING***.

   At this point, you can connect to cluster instances, and the cluster sequentially runs any steps that you specified when you created the cluster. You can submit additional steps, which run after any previous steps complete.

5. After steps run successfully, the cluster goes into a ***WAITING*** state.

   If a cluster is configured to auto-terminate after the last step is complete, it goes into a ***TERMINATING*** state and then into the ***TERMINATED*** state.

   Whereas if the cluster is configured to wait, you must manually shut it down when you no longer need it. After you manually shut down the cluster, it goes into the ***TERMINATING*** state and then into the ***TERMINATED*** state.

But what about failure??

A failure during the cluster lifecycle causes Amazon EMR to terminate the cluster and all of its instances unless you enable termination protection. If a cluster terminates because of a failure, any data stored on the cluster is deleted, and the cluster state is set to ***TERMINATED_WITH_ERRORS***. If you enabled termination protection, you can retrieve data from your cluster, and then remove termination protection and terminate the cluster

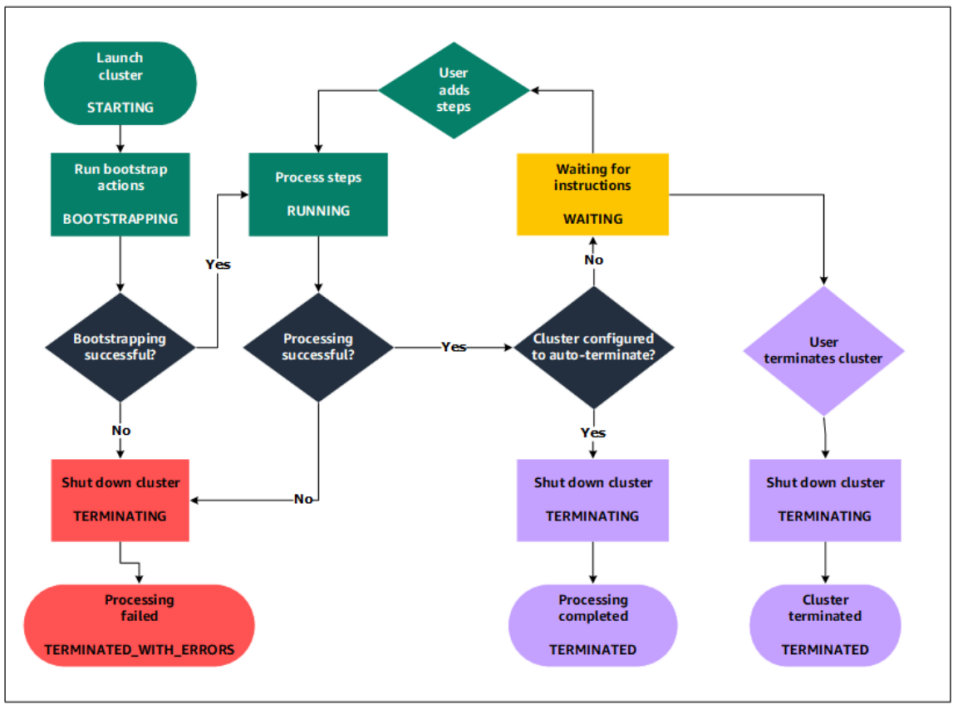

Figure 6.1: EMR lifecycle flow chart