Introduction to YARN

Adam Kawa (kawa.adam@gmail.com)
Hadoop Developer, Administrator, and Instructor
Spotify and GetInData

12 August 2014
(First published 01 April 2014)

Apache Hadoop is one of the most popular tools for big data processing. It has been successfully deployed in production by many companies for several years. Though Hadoop is considered a reliable, scalable, and cost-effective solution, it is constantly being improved by a large community of developers. As a result, the 2.0 version offers several revolutionary features, including Yet Another Resource Negotiator (YARN), HDFS Federation, and a highly available NameNode, which make the Hadoop cluster much more efficient, powerful, and reliable. In this article, learn about the advantages YARN provides over the previous version of the distributed processing layer in Hadoop.

Introduction

Apache Hadoop 2.0 includes YARN, which separates the resource management and processing components. The YARN-based architecture is not constrained to MapReduce. This article describes YARN and its advantages over the previous distributed processing layer in Hadoop. Learn how to enhance your clusters with YARN's scalability, efficiency, and flexibility.

Apache Hadoop in a nutshell

Apache Hadoop is an open source software framework that can be installed on a cluster of commodity machines so the machines can communicate and work together to store and process large amounts of data in a highly distributed manner. Initially, Hadoop consisted of two main components: a Hadoop Distributed File System (HDFS) and a distributed computing engine that lets you implement and run programs as MapReduce jobs.

MapReduce, a simple programming model popularized by Google, is useful for processing large datasets in a highly parallel and scalable way. MapReduce is inspired by functional programming whereby users express their computation as map and reduce functions that process data as key-value pairs. Hadoop provides a high-level API for implementing custom map and reduce functions in various languages.

Hadoop also provides the software infrastructure for running MapReduce jobs as a series of map and reduce tasks. *Map tasks* invoke map functions over subsets of input data. After they are done, *reduce tasks* start calling reduce functions on the intermediate data, generated by map functions, to produce the final output. Map and reduce tasks run in isolation from one another, which allows for a parallel and fault-tolerant computation.

Most importantly, the Hadoop infrastructure takes care of all complex aspects of distributed processing: parallelization, scheduling, resource management, inter-machine communication, handling software and hardware failures, and more. Thanks to this clean abstraction, implementing distributed applications that process terabytes of data on hundreds (or even thousands) of machines has never been so easy — even for developers with no previous experience with distributed systems.

Hadoop's golden era

Although there were several open source implementations of the MapReduce model, Hadoop MapReduce quickly became the most popular. Hadoop is also one of the most exciting open source projects on the planet, thanks to several great features: a high-level API, near-linear scalability, an open source licence, the ability to run on commodity hardware, and fault tolerance. It has been successfully deployed by hundreds (maybe thousands) of companies and is the current standard for large-scale distributed storage and processing.

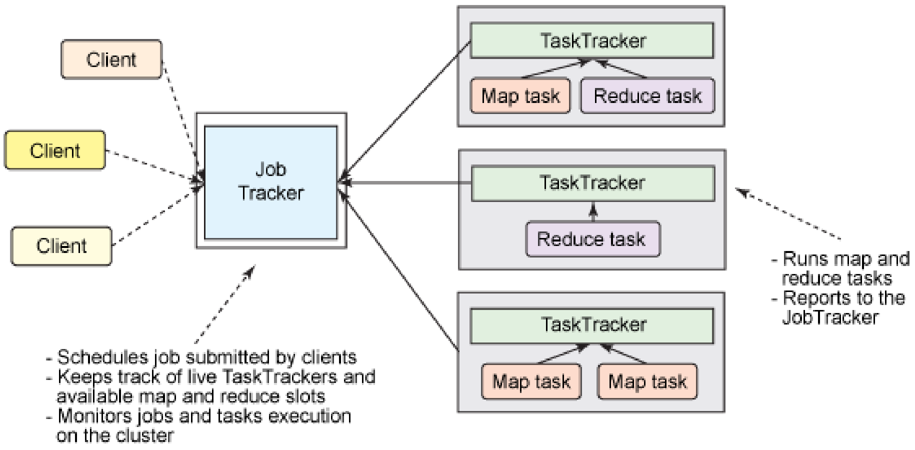
Some early Hadoop adopters, such as Yahoo! and Facebook, have built large clusters in the range of 4,000 nodes to satisfy their constantly growing and varying data processing needs. After they built their cluster, though, they started noticing limitations of the Hadoop MapReduce framework.

Limitations of classical MapReduce

The most serious limitations of classical MapReduce are primarily related to scalability, resource utilization, and the support of workloads different from MapReduce. In the MapReduce framework, the job execution is controlled by two types of processes:

- A single master process called *JobTracker*, which coordinates all jobs running on the cluster and assigns map and reduce tasks to run on the TaskTrackers
- A number of subordinate processes called *TaskTrackers*, which run assigned tasks and periodically report the progress to the JobTracker

Classical version of Apache Hadoop (MRv1)


Client
Client
Client
Job Tracker
TaskTracker
Map task
Reduce task
TaskTracker
Reduce task
TaskTracker
Map task
Map task
- Runs map and reduce tasks
- Reports to the JobTracker
- Schedules job submitted by clients
- Keeps track of live TaskTrackers and available map and reduce slots
- Monitors jobs and tasks execution on the cluster

The large Hadoop clusters revealed a limitation involving a scalability bottleneck caused by having a single JobTracker. According to Yahoo!, the practical limits of such a design are reached with a cluster of 5,000 nodes and 40,000 tasks running concurrently. Due to this limitation, smaller and less-powerful clusters had to be created and maintained.

Moreover, both smaller and larger Hadoop clusters had never used their computational resources with optimum efficiency. In Hadoop MapReduce, the computational resources on each slave node are divided by a cluster administrator into a fixed number of map and reduce slots, which are not fungible. With the number of map and reduce slots set, a node cannot run more map tasks than map slots at any given moment, even if no reduce tasks are running. It harms the cluster utilization because when all map slots are taken (and we still want more), we cannot use any reduce slots, even if they are available, or vice versa.

Last, but not least, Hadoop was designed to run MapReduce jobs only. With the advent of alternative programming models (such as graph processing provided by Apache Giraph), there was an increasing need to support programming paradigms besides MapReduce that could run on the same cluster and share resources in an efficient and fair manner.

In 2010, engineers at Yahoo! began working on a completely new architecture of Hadoop that addresses all the limitations above and multiple additional features.

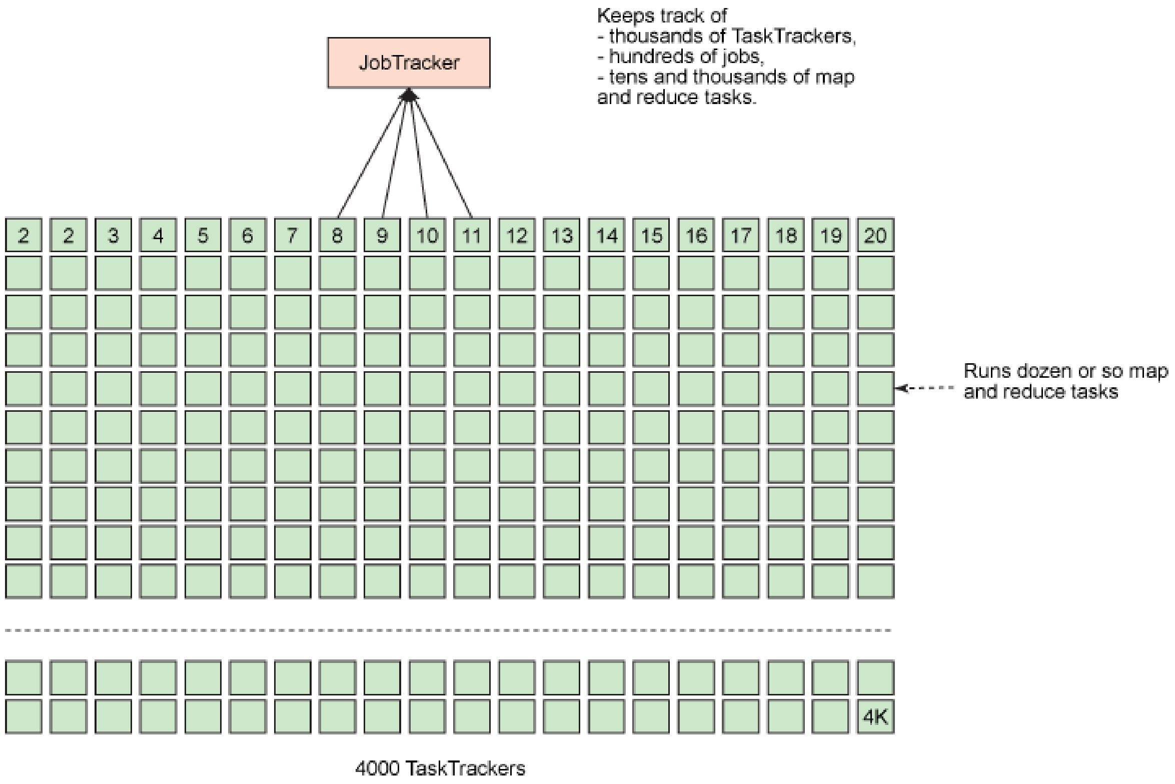
Addressing the scalability issue

In Hadoop MapReduce, the JobTracker is charged with two distinct responsibilities:

- Management of computational resources in the cluster, which involves maintaining the list of live nodes, the list of available and occupied map and reduce slots, and allocating the available slots to appropriate jobs and tasks according to selected scheduling policy
- Coordination of all tasks running on a cluster, which involves instructing TaskTrackers to start map and reduce tasks, monitoring the execution of the tasks, restarting failed tasks, speculatively running slow tasks, calculating total values of job counters, and more

The large number of responsibilities given to a single process caused significant scalability issues, especially on a larger cluster where the JobTracker had to constantly keep track of thousands of TaskTrackers, hundreds of jobs, and tens of thousands of map and reduce tasks. The image below illustrates the issue. On the contrary, the TaskTrackers usually run only a dozen or so tasks, which were assigned to them by the hard-working JobTracker.

Busy JobTracker on a large Apache Hadoop cluster (MRv1)


JobTracker
Keeps track of
- thousands of TaskTrackers,
- hundreds of jobs,
- tens and thousands of map and reduce tasks.
2
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Runs dozen or so map and reduce tasks
4K
4000 TaskTrackers

To address the scalability issue, a simple but brilliant idea was proposed: Let's somehow reduce the responsibilities of the single JobTracker and delegate some of them to the TaskTrackers since there are many of them in a cluster. This concept was reflected in a new design by separating dual responsibilities of the JobTracker (cluster resource management and task coordination) into two distinct types of processes.

Instead of having a single JobTracker, a new approach introduces a cluster manager with the sole responsibility of tracking live nodes and available resources in the cluster and assigning them to the tasks. For each job submitted to a cluster, a dedicated and short-living JobTracker is started to control the execution of tasks within that job only. Interestingly, the short-living JobTrackers are started by the TaskTrackers running on slave nodes. Thus, the coordination of a job's life cycle is spread across all of the available machines in the cluster. Thanks to this behavior, more jobs can run in parallel and scalability is dramatically increased.

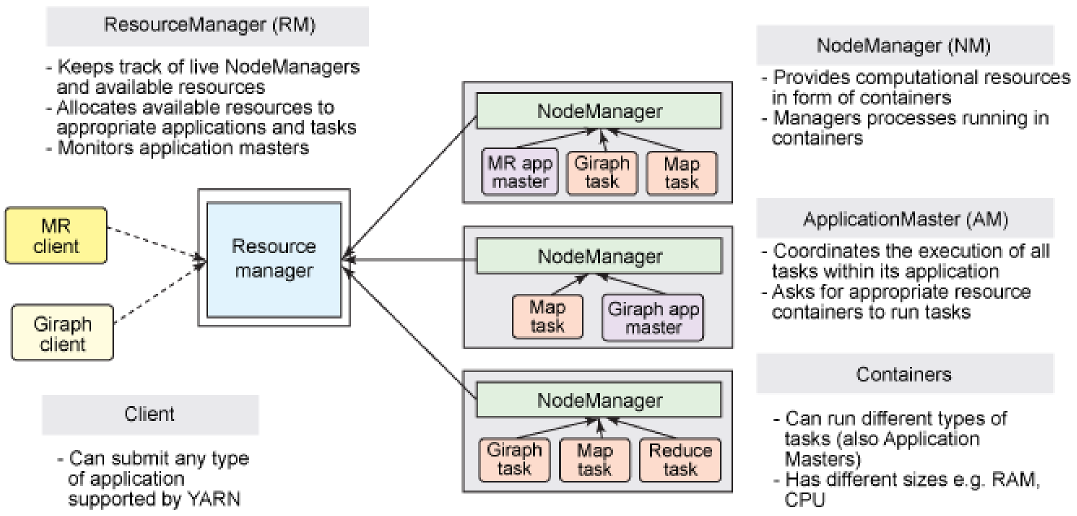
YARN: The next generation of Hadoop's compute platform

Let's slightly change the terminology now. The following name changes give a bit of insight into the design of YARN:

- ResourceManager instead of a cluster manager
- ApplicationMaster instead of a dedicated and short-lived JobTracker
- NodeManager instead of TaskTracker
- A distributed application instead of a MapReduce job

YARN is the next generation of Hadoop's compute platform, as shown below.

Architecture of YARN


ResourceManager (RM)
- Keeps track of live NodeManagers and available resources
- Allocates available resources to appropriate applications and tasks
- Monitors application masters
MR client
Giraph client
Resource manager
Client
- Can submit any type of application supported by YARN
NodeManager
MR app master
Giraph task
Map task
NodeManager
Map task
Giraph app master
NodeManager
Giraph task
Map task
Reduce task
NodeManager (NM)
- Provides computational resources in form of containers
- Managers processes running in containers
ApplicationMaster (AM)
- Coordinates the execution of all tasks within its application
- Asks for appropriate resource containers to run tasks
Containers
- Can run different types of tasks (also Application Masters)
- Has different sizes e.g. RAM, CPU

In the YARN architecture, a global ResourceManager runs as a master daemon, usually on a dedicated machine, that arbitrates the available cluster resources among various competing applications. The ResourceManager tracks how many live nodes and resources are available on the cluster and coordinates what applications submitted by users should get these resources and when. The ResourceManager is the single process that has this information so it can make its allocation (or rather, scheduling) decisions in a shared, secure, and multi-tenant manner (for instance, according to an application priority, a queue capacity, ACLs, data locality, etc.).

When a user submits an application, an instance of a lightweight process called the ApplicationMaster is started to coordinate the execution of all tasks within the application. This includes monitoring tasks, restarting failed tasks, speculatively running slow tasks, and calculating total values of application counters. These responsibilities were previously assigned to the single JobTracker for all jobs. The ApplicationMaster and tasks that belong to its application run in resource containers controlled by the NodeManagers.

The NodeManager is a more generic and efficient version of the TaskTracker. Instead of having a fixed number of map and reduce slots, the NodeManager has a number of dynamically created resource containers. The size of a container depends upon the amount of resources it contains, such as memory, CPU, disk, and network IO. Currently, only memory and CPU (YARN-3) are supported. cgroups might be used to control disk and network IO in the future. The number of containers on a node is a product of configuration parameters and the total amount of node resources (such as total CPUs and total memory) outside the resources dedicated to the slave daemons and the OS.

Interestingly, the ApplicationMaster can run any type of task inside a container. For example, the MapReduce ApplicationMaster requests a container to launch a map or a reduce task, while the Giraph ApplicationMaster requests a container to run a Giraph task. You can also implement a custom ApplicationMaster that runs specific tasks and, in this way, invent a shiny new distributed application framework that changes the big data world. I encourage you to read about Apache Twill, which aims to make it easy to write distributed applications sitting on top of YARN.

In YARN, MapReduce is simply degraded to a role of a distributed application (but still a very popular and useful one) and is now called MRv2. MRv2 is simply the re-implementation of the classical MapReduce engine, now called MRv1, that runs on top of YARN.

One cluster that can run any distributed application

The ResourceManager, the NodeManager, and a container are not concerned about the type of application or task. All application framework-specific code is simply moved to its ApplicationMaster so that any distributed framework can be supported by YARN — as long as someone implements an appropriate ApplicationMaster for it.

Thanks to this generic approach, the dream of a Hadoop YARN cluster running many various workloads comes true. Imagine: a single Hadoop cluster in your data center that can run MapReduce, Giraph, Storm, Spark, Tez/Impala, MPI, and more.

The single-cluster approach obviously provides a number of advantages, including:

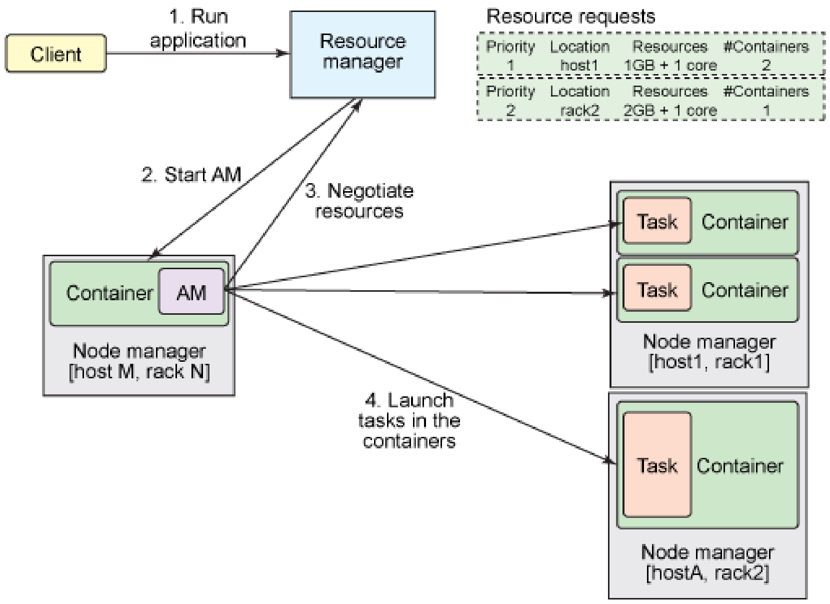
- Higher cluster utilization, whereby resources not used by one framework could be consumed by another
- Lower operational costs, because only one "do-it-all" cluster needs to be managed and tuned
- Reduced data motion, as there's no need to move data between Hadoop YARN and systems running on different clusters of machines

Managing a single cluster also results in a greener solution to data processing. Less data center space is used, less silicon wasted, less power used, and less carbon emitted simply because we run the same calculation on a smaller but more efficient Hadoop cluster.

Application submission in YARN

This section discusses how the ResourceManager, ApplicationMaster, NodeManagers, and containers interact together when an application is submitted to a YARN cluster. The image below shows an example.

Application submission in YARN


Client
1. Run application
Resource manager
Resource requests
Priority 1 Location host1 Resources 1GB + 1 core #Containers 2
Priority 2 Location rack2 Resources 2GB + 1 core #Containers 1
2. Start AM
3. Negotiate resources
Container AM
Node manager [host M, rack N]
Task Container
Task Container
Node manager [host1, rack1]
4. Launch tasks in the containers
Task Container
Node manager [hostA, rack2]

Suppose that users submit applications to the ResourceManager by typing the `hadoop jar` command in the same manner as in MRv1. The ResourceManager maintains the list of applications running on the cluster and the list of available resources on each live NodeManager. The ResourceManager needs to determine which application should get a portion of cluster resources next. The decision is subjected to many constraints, such as queue capacity, ACLs, and fairness. The ResourceManager uses a pluggable Scheduler. The Scheduler focuses only on scheduling; it manages who gets cluster resources (in the form of containers) and when, but it does not perform any monitoring of the tasks within an application so it does not attempt to restart failed tasks.

When the ResourceManager accepts a new application submission, one of the first decisions the Scheduler makes is selecting a container in which ApplicationMaster will run. After the ApplicationMaster is started, it will be responsible for a whole life cycle of this application. First and foremost, it will be sending resource requests to the ResourceManager to ask for containers needed to run an application's tasks. A resource request is simply a request for a number of containers that satisfies some resource requirements, such as:

- An amount of resources, today expressed as megabytes of memory and CPU shares
- A preferred location, specified by hostname, rackname, or * to indicate no preference
- A priority within this application, and not across multiple applications

If and when it is possible, the ResourceManager grants a container (expressed as container ID and hostname) that satisfies the requirements requested by the ApplicationMaster in the resource request. A container allows an application to use a given amount of resources on a specific host. After a container is granted, the ApplicationMaster will ask the NodeManager (that manages the host on which the container was allocated) to use these resources to launch an application-specific task. This task can be any process written in any framework (such as a MapReduce task or a Giraph task). The NodeManager does not monitor tasks; it only monitors the resource usage in the containers and, for example, it kills a container if it consumes more memory than initially allocated.

The ApplicationMaster spends its whole life negotiating containers to launch all of the tasks needed to complete its application. It also monitors the progress of an application and its tasks, restarts failed tasks in newly requested containers, and reports progress back to the client that submitted the application. After the application is complete, the ApplicationMaster shuts itself down and releases its own container.

Though the ResourceManager does not perform any monitoring of the tasks within an application, it checks the health of the ApplicationMasters. If the ApplicationMaster fails, it can be restarted by the ResourceManager in a new container. You can say that the ResourceManager takes care of the ApplicationMasters, while the ApplicationMasters takes care of tasks.

Interesting facts and features

YARN offers several other great features. Describing all of them is outside the scope of this article, but I've included some noteworthy features:

- *Uberization* is the possibility to run all tasks of a MapReduce job in the ApplicationMaster's JVM if the job is small enough. This way, you avoid the overhead of requesting containers from the ResourceManager and asking the NodeManagers to start (supposedly small) tasks.
- Binary or source compatibility for MapReduce jobs written for MRv1 (MAPREDUCE-5108).
- High availability for the ResourceManager (YARN-149). This work is in progress, and is already done by some vendors.
- An application recovery after the restart of ResourceManager (YARN-128). The ResourceManager stores information about running applications and completed tasks in HDFS. If the ResourceManager is restarted, it recreates the state of applications and re-runs only incomplete tasks. This work is close to completion and has been actively tested by the community. It is already done by some vendors.
- Simplified user-log management and access. Logs generated by applications are not left on individual slave nodes (as with MRv1) but are moved to a central storage, such as HDFS. Later, they can be used for debugging purposes or for historical analyses to discover performance issues.
- A new look and feel of the web interface.

Conclusion

YARN is a completely rewritten architecture of Hadoop cluster. It seems to be a game-changer for the way distributed applications are implemented and executed on a cluster of commodity machines.

YARN offers clear advantages in scalability, efficiency, and flexibility compared to the classical MapReduce engine in the first version of Hadoop. Both small and large Hadoop clusters greatly benefit from YARN. To the end user (a developer, not an administrator), the changes are almost invisible because it's possible to run unmodified MapReduce jobs using the same MapReduce API and CLI.

There is no reason not to migrate from MRv1 to YARN. The biggest Hadoop vendors agree on that point and offer extensive support for running Hadoop YARN. Today, YARN is successfully used in production by many companies, such as Yahoo!, eBay, Spotify, Xing, Allegro, and more.

Acknowledgements

I would like to thank Piotr Krewski and Fabian Alenius for their technical review of this article.

Resources

Learn

- Learn about the IBM Watson research project.
- Check out Big Data University for free courses on Hadoop and big data.
- Read *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data* for details on two of IBM's key big data technologies.
- Visit the Apache Hadoop Project to learn what it is, how to get it, how to get started, and all the news.
- Read "The Hadoop Distributed File System: Architecture and Design," by Dhruba Borthakur.
- Learn about HadoopDB, an architectural hybrid of MapReduce and DBMS technologies for analytical workloads.
- Read the Hadoop MapReduce tutorial at Apache.org.
- Read "Using MapReduce and load balancing on the cloud" to learn how to implement the Hadoop MapReduce framework in a cloud environment and how to use virtual load balancing to improve the performance of both a single- and multiple-node system.
- For information about installing Hadoop using CDH4, see CDH4 Installation — Cloudera Support.
- Use the *Big Data Glossary* (Pete Warden, O'Reilly Media, ISBN: 1449314597, 2011) for help navigating new data tools and innovations.
- *Hadoop: The Definitive Guide* (Tom White, O'Reilly Media, ISBN: 1449389732, 2010) explains how to build and maintain reliable, scalable, distributed systems with the Hadoop framework.
- HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads (Azza Abouzeid et al., Proceedings of the VLDB Endowment, 2(1), 2009) explores the feasibility of building a hybrid system that takes the best features from both technologies.
- Read MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI, 2004.
- SQL/MapReduce: A practical approach to self-describing, polymorphic, and parallelizable user-defined functions (Friedman et al., Proceedings of the VLDB Endowment, 2(2), 2009) describes the motivation for this new approach to UDFs as well as the implementation within AsterData Systems' nCluster database.
- "MapReduce and parallel DBMSes: friends or foes?" (Stonebraker et al., Commun. ACM 53(1), 2010) argues that using MR systems to perform tasks best suited for DBMSes yields less-than-satisfactory results.
- A Survey of Large Scale Data Management Approaches in Cloud Environments (Sakr et al., Journal of IEEE Communications Surveys and Tutorials, 13(3), 2011) gives a comprehensive survey of numerous approaches and mechanisms of deploying data-intensive applications in the cloud.
- Visit the developerWorks Open source zone to find extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM products.

Get products and technologies

- Refer to the InfoSphere BigInsights Information Center for product documentation.
- Get Hadoop 0.20.1, Hadoop MapReduce, and Hadoop HDFS from Apache.org.
- Innovate your next development project with IBM trial software, available for download or on DVD.

Discuss

- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Adam Kawa

Adam Kawa works as a data engineer at Spotify, where his main responsibility is to maintain one of the largest Hadoop-YARN clusters in Europe. Every so often, he implements and troubleshoots MapReduce, Hive, Tez, and Pig applications. Adam also works as a Hadoop instructor at GetInData and is a frequent speaker at Hadoop conferences and Hadoop User Groups meetups. He co-organizes Stockholm and Warsaw Hadoop User Groups. Adam regularly blogs about the Hadoop ecosystem at HakunaMapData.com.