Big Data Technologies

Editor: Gregor von Laszewski

March 25, 2017

0.1 Contributors

Name	**HID**	**Title**	**Pages**
Avadhoot Agasti	SL-IO-3000	TBD	1
Abhishek Gupta	S17-IO-3005	TBD	1
Matthew Lawson	S17-IO-3010	TBD	1
Nandita Sathe	S17-IO-3017	TBD	1
Pratik Jain	S17-IR-2012	TBD	1
Kumar Satyam	S17-IR-2031	TBD	1

Apache Ranger

AVADHOOT AGASTI[1,*,+]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: aagasti@indiana.edu
[+] HID - SL-IO-3000

paper2, March 25, 2017

Apache Hadoop provides various data storage, data access and data processing services. Apache Ranger is part of the Hadoop eco-system. Apache Ranger provides capability to perform security administration tasks for storage, access and processing of data in Hadoop. Using Ranger, Hadoop administrator can perform security administration tasks using a central UI or Restful web services. He can define policies which enable users/user-groups to perform specific action using Hadoop components and tools. Ranger provides role based access control for datasets on Hadoop at column and row level. Ranger also provides centralized auditing of user access and security related administrative actions.

Keywords: Apache Ranger, LDAP, Active Directory, Apache Knox, Apache Atlas, Apache Hive, Apache Hadoop, Yarn, Apache HBase, Apache Storm, Apache Kafka, Data Lake, Apache Sentry

https://github.com/avadhoot-agasti/sp17-i524/tree/master/paper2/S17-IO-3000/report.pdf

1. INTRODUCTION

Apache Ranger is open source software project designed to provide centralized security services to various components of Apache Hadoop. Apache Hadoop provides various mechanism to store, process and access the data. Each Apache tool has its own security mechanism. This increases administrative overhead and is also error prone. Apache Ranger fills this gap to provide a central security and auditing mechanism for various Hadoop components. Using Ranger, Hadoop administrator can perform security administration tasks using a central UI or Restful web services. He can define policies which enable users/user-groups to perform specific action using Hadoop components and tools. Ranger provides role based access control for datasets on Hadoop at column and row level. Ranger also provides centralized auditing of user access and security related administrative actions.

2. ARCHITECTURE OVERVIEW

[1] describes the important components of Ranger as explained below:

2.1. Ranger Admin Portal

Ranger Admin Portal is the main interaction point for the user. Using Admin Portal, user can define policies. The policies are stored in Policy Database. The Policies are polled by various plugins. The Admin Portal also collects the audit data from plugins and stores in HDFS or in a relational database.

2.2. Ranger Plugins

Plugins are Java Programs, which are invoked as part of the cluster component. For example, the ranger-hive plugin is embedded as part of Hive Server2. The plugins cache the policies, and intercept the user request and evaluates it against the policies. Plugins also collect the audit data for that specific component and send to Admin Portal.

2.3. User group sync

While Ranger provides authorization or access control mechanism, it needs to know the users and the groups. Ranger integrates with Unix users management or LDAP or Active Directory to fetch the users and groups. The User group sync component is responsible for this integration.

3. HADOOP COMPONENTS SUPPORTED BY RANGER

Ranger supports auditing and authorization for following Hadoop components [2].

3.1. Apache Hadoop and HDFS

Apache Ranger provides plugin for Hadoop, which helps in enforcing data access policies. The HDFS plugin works with Name Node to check if the user's access request to a file on HDFS is valid or not.

3.2. Apache Hive

Apache Hive provides SQL interface on top of the data stored in HDFS. Apache Hive supports two types of authorization -

Storage based authorization and SQL standard authorization. Ranger provides centralized authorization interface for Hive which provides granular access control at table and column level. Ranger implements a plugin which is part of Hive Server2.

3.3. Apache HBase

Apache HBase is NoSQL database implemented on top of Hadoop and HDFS. Ranger provides coprocessor plugin for HBase, which performs authorization checks and audit log collections.

3.4. Apache Storm

Ranger provides plugin to Nimbus server which helps in performing the security authorization on Apache Storm.

3.5. Apache Knox

Apache Knox provides service level authorization for users and groups. Ranger provides plugin for Knox using which, administration of policies can be supported. The audit over Knox data enables user to perform detailed analysis of who and when accessed Knox.

3.6. Apache Solr

Solr provides free text search capabilities on top of Hadoop. Ranger is useful to protect Solr collections from unauthorized usage.

3.7. Apache kafka

Ranger can manager access control on Kafka topics. Policies can be implemented to control which users can write to a Kafka topic and which users can read from a Kafka topic.

3.8. Yarn

Yarn is resource management layer for Hadoop. Administrators can setup queues in Yarn and then allocate users and resources per queue basis. Policies can be defined in Ranger to define who can write to various Yarn queues.

4. IMPORTANT FEATURES OF RANGER

The blog article [3] explains the 2 important features of Apache Ranger.

4.1. Dynamic Column Masking

Dynamic data masking at column level is an important feature of Apache Ranger . Using this feature, the administrator can setup data masking policy. The data masking makes sure that only authorized users can see the actual data while other users will see the masked data. Since the masked data is format preserving, they can continue their work without getting access to the actual sensitive data. For example, the application developers can use masked data to develop the application whereas when the application is actually deployed, it will show actual data to the authorized user. Similarly, a security administrator may chose to mask credit card number when it is displayed to a service agent.

4.2. Row Level Filtering

The data authorization is typically required at column level as well as at row level. For example, in an organization which is geographically distributed in many locations, the security administrator may want to give access of a data from a specific location to the specific user. In other example, a hospital data security administrator may want to allow doctors to see only his or her patients. Using Ranger, such row level access control can be specified and implemented.

5. HADOOP DISTRIBUTION SUPPORT

Ranger can be deployed on top of Apache Hadoop. [4] provides detailed steps of building and deploying Ranger on top of Apache Hadoop.

Hortonwork Distribution of Hadoop (HDP) supports Ranger deployment using Ambari. [5] provides installation, deployment and configuration steps for Ranger as part of HDP deployment.

Cloudera Hadoop Distribution (CDH) does not support Ranger. According to [6], Ranger is not recommended on CDH and instead Apache Sentry should be used as central security and audit tool on top of CDH.

6. USE CASES

Apache Ranger provides centralized security framework which can be useful in many use cases as explained below.

6.1. Data Lake

[7] explains that storing many types of data in the same repository is one of the most important feature of Data Lake. With multiple datasets, the ownership, security and access control of the data becomes primary concern. Using Apache Ranger, the security administrator can define fine grain control on the data access.

6.2. Multi-tenant Deployment of Hadoop

Hadoop provides ability to store and process data from multiple tenants. The security framework provided by Apache Ranger can be utilized to protect the data and resources from un-authorized access.

7. APACHE RANGER AND APACHE SENTRY

According to [8], Apache Sentry and Apache Ranger have many features in common. Apache Sentry ([9]) provides role based authorization to data and metadata stored in Hadoop.

8. EDUCATIONAL MATERIAL

[10] provides tutorial on topics like A)Security resources B)Auditing C)Securing HDFS, Hive and HBase with Knox and Ranger D) Using Apache Atlas' Tag based policies with Ranger. [11] provides step by step guidance on getting latest code base of Apache Ranger, building and deploying it.

9. LICENSING

Apache Ranger is available under Apache 2.0 License.

10. CONCLUSION

Apache Ranger is useful to Hadoop Security Administrators since it enables the granular authorization and access control. It also provides central security framework to different data storage and access mechanism like Hive, HBase and Storm. Apache Ranger also provides audit mechanism. With Apache Ranger, the security can be enhanced for complex Hadoop use cases like Data Lake.

ACKNOWLEDGEMENTS

The authors thank Prof. Gregor von Laszewski for his technical guidance.

REFERENCES

[1] Hortonworks, “Apache ranger - overview,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: https://hortonworks.com/apache/ranger/#section_2

[2] A. S. Foundation, “Apache ranger - frequenty asked questions,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: http://ranger.apache.org/faq.html#How_does_it_work_over_Hadoop_and_related_components

[3] S. Mahmood and S. Venkat, “For your eyes only: Dynamic column masking & row-level filtering in hdp2.5,” Web Page, Sep. 2016, online; accessed 9-Mar-2017. [Online]. Available: https://hortonworks.com/blog/eyes-dynamic-column-masking-row-level-filtering-hdp2-5/

[4] A. S. Foundation, “Apache ranger 0.5.0 installation,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: https://cwiki.apache.org/confluence/display/RANGER/Apache+Ranger+0.5.0+Installation

[5] Horto, “Installing apache rang,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.6/bk_installing_manually_book/content/ch_installing_ranger_chapter.html

[6] Cloudera, “Configuring authorization,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: https://www.cloudera.com/documentation/enterprise/5-6-x/topics/sg_authorization.html

[7] Teradata and Hortonworks, “Putting the data lake to work - a guide to best practices,” Web Page, Apr. 2014, online; accessed 9-Mar-2017. [Online]. Available: https://hortonworks.com/wp-content/uploads/2014/05/TeradataHortonworks_Datalake_White-Paper_20140410.pdf

[8] S. Neumann, “5 hadoop security projects,” Web Page, Nov. 2014, online; accessed 9-Mar-2017. [Online]. Available: https://www.xplenty.com/blog/2014/11/5-hadoop-security-projects/

[9] A. S. Foundation, “Apache sentry,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: https://sentry.apache.org/

[10] Hortonworks, “Apache ranger overview,” Web, online; accessed 9-Mar-2017. [Online]. Available: https://hortonworks.com/apache/ranger/#tutorials

[11] A. S. Foundation, “Apache ranger - quick start guide,” Web Page, online; accessed 9-Mar-2017. [Online]. Available: http://ranger.apache.org/quick_start_guide.html

Amazon Kinesis

ABHISHEK GUPTA[1,*]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: abhigupt@iu.edu*

project-001, March 25, 2017

Amazon Kinesis [1] provides a software-as-a-service(SAAS) platform to application developers on Amazon Web Services(AWS) platform, which is capable of processing streaming data at in real time. This is a key challenge application developers face when they have to process huge amounts of data in real time. It can scale up or scale down based on data needs of the system. As volume of data grows with advent IOT devices and sensors, Kinesis will play a key role in developing applications which require insights in real time with this growing volume of data.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/paper2/S17-IO-3005/report.pdf

1. INTRODUCTION

Amazon Kinesis [1] gives application developers collect and analyze streaming data in real time. The stream data can come from variety of sources like social media, sensors, mobile devices, syslogs, logs, web server logs, network data etc. Kinesis can scale on demand as application needs changes. For example during peak load situation kinesis added more workers nodes and can reduce the nodes when the application runs at low load. It also provides durability, where if streams nodes go down the data is persisted on disk and get replicated when new nodes come up. Multiple applications can consume data from one or more streams for variety of use cases for example one application computes moving average and another application counts the number of users clicks. These applications can work in parallel and independently. Kinesis provides streaming in realtime with sub second delays between producer and consumer. Kinesis has two type of processing engines

- Kinesis streams - reads data from producers
- Kinesis firehose - pushes data to consumers

Kinesis streams can used to process incoming data processing from multiple datasources while firehose is used to load streaming data into aws like Kinesis analytics, S3, Redshift, Elasticsearch etc.

2. ARCHITECTURE

Amazon Kinesis reads data from variety of sources. The data coming into streams is in a record format. Each record is composed on a partition key, sequence number and data blob which is raw serialized byte array. The data further is partitioned into multiple workers(or shards) using the incoming partition key.

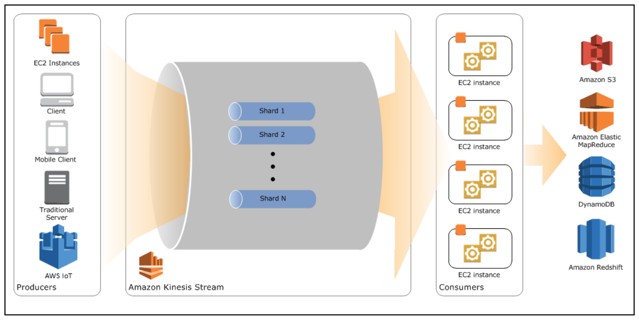


Fig. 1. Kinesis streams building blocks
[2]

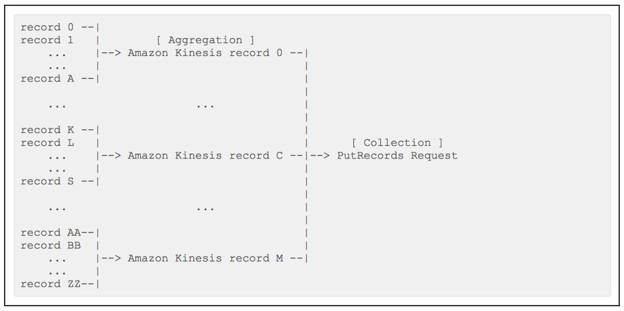
Following are key components in streams architecture [2] :

2.1. Data Record

Its one unit of data that flows through Kinesis stream. Data records is made up of sequence number, partition key, and blob of actual data. Size of data blob is max 1 MB. During aggregation one more records are aggregated in to a single aggregated record. Further these aggregated records are emitted as an aggregation collection.

2.2. Producer

Producers write the data to Kinesis stream. Producer can be any system producing data for example ad server, social media stream, log server etc,



```
record 0 --|
record 1   |        [ Aggregation ]
    ...    |--> Amazon Kinesis record 0 --|
    ...    |                              |
record A --|                              |
                                          |
    ...                   ...             |
                                          |
record K --|                              |
record L   |                              |      [ Collection ]
    ...    |--> Amazon Kinesis record C --|--> PutRecords Request
    ...    |                              |
record S --|                              |
                                          |
    ...                   ...             |
                                          |
record AA--|                              |
record BB  |                              |
    ...    |--> Amazon Kinesis record M --|
    ...    |
record ZZ--|
```

Fig. 2. Aggregation of records

2.3. Consumer

Consumers subscribe to one or more streams. Consumer can be one of the application running on aws or hosted on EC2.

2.4. Consumer

A stream can have one or more shards. Records are processed by each shard based on the partition key. Each shard can process upto 2MB/s data for reads and upto 1MB/s for writes. Total capacity of a stream is sum of capacities of its shards.

2.5. Partition Keys

Partition key is 256 bytes long. A MD5 hash function is used to map partition keys to 128 bit integer values which is further used to map to appropriate shard.

2.6. Sequence Number

Sequence number is assigned to a record when a record get written to the stream.

2.7. Amazon Kinesis Client Library

Amazon Kinesis Client Library is bundled into your application built on AWS. It makes sure for each record there is a shard available to process that record. Client library uses dynamo db to store control data records being processed by shards.

2.8. Application Name

Name of application in the control table in dynamodb where kinesis streams will write the data to. This name is unique.

3. CREATING STREAMS

You can create a stream using following ways:

- kinesis console
- Streams API
- AWS CLI

Before creating stream you should determine initial size of the stream [3] and number of shards required to create your stream. Number of shards can be calculated using the following formulae

$$
\begin{aligned}
number_of_shards = \\
&max(incomingWriteBandwidthInKB/1000, \\
&\qquad outgoingReadBandwidthInKB/2000)
\end{aligned}
$$

Here, the attributes used in the calculation are self explanatory.

Producer for streams writes data records into Kinesis streams. This data is available for 24 hours within streams. The default retention interval can be changed. To write records to stream, you must specify partition key, name of stream and data blob. Consumer on the other hand reads data from streams using shard iterator. Shard iterator provides consumer a position on streams from where the consumer can start reading the data.

4. STREAM LIMITS

4.1. Shard

By default there can 25 shards in a region except US east, EU and US west which has limit of 50 shards. Each shard can support up to 5 transactions per second for reads and maximum data rate of 2 MB per second. Each shard can support 1000 records per second for writes and maximum data rate of 1MB per second.

4.2. Data retention

By default the data is available for 24 hours which can be configured up to 168 hours with 1 hour increments.

4.3. Data Blob

Maximum size of data blob is 1MB before base64 encoding.

5. MANAGEMENT

Kinesis provides all management using:

- AWS console
- Java SDK [4]

5.1. Java SDK

AWS provides a java SDK. Java SDK [4] can be used to complete all workflow on stream. Workflow like create, listing, retrieving shards from stream, deleting stream, resharding stream and changing data retention period. SDK provide rich documentation and developer blogs to support development on streams.

5.2. AWS console

AWS provides a web console to manage all AWS services including kinesis. Using console web user interface user can perform all operations to manage stream.

6. MONITORING

AWS provides several ways to monitor streams. These are:

- CloudWatch metrics
- Kinesis Agent
- API logging
- Client library
- Producer Library

Using CloudWatch metrics allows you can monitor the data and usage at shard level. It can collect metrics like: latency, coming bytes, incoming records, success count etc.

7. LICENSING

Kinesis is software as a service(SAAS) from Amazon AWS infrastructure. Hence it can only run as a service within AWS. It comes with pay as you go pricing.

8. USE CASES

Kinesis streams and firehose can be useful in variety of use cases [1]

- log data processing
- log mining
- realtime metrics and reporting
- realtime analytics
- complex stream processing

Kinesis solves variety of these business problems by doing a real time analysis and aggregation. This aggregated data can further be stored or available to query. Since it runs on amazon, it becomes easy for users to integrate and use other AWS components.

9. CONCLUSION

Kinesis can process huge amounts of data in realtime. Application developers can then focus on business logic. Kinesis [5] an help build realtime dashboards, capture anomalies, generate alerts, provide recommendations which can help take business and operation decisions in real time. It can also send data to other AWS services like S3, dynamodb, redshift etc. You can scale up or scale down as you demand increases or decreases and pay based on your usage. Only downside of Kinesis it that it cannot run on a private or hybrid cloud rather can only on AWS public cloud or Amazon VPC(Virtual Private Cloud). Customers who want to use Kinesis but don't want to be on Amazon platform cannot use it. Its comparable with open source Apache Kafka[6] . However, Kafka lacks in high availability and monitoring in case of cloud deployments.

ACKNOWLEDGEMENTS

Special thanks to Professor Gregor von Laszewski, Dimitar Nikolov and all associate instructors for all help and guidance related to latex and bibtex, scripts for building the project, quick and timely resolution to any technical issues faced. The paper is written during the course I524: Big Data and Open Source Software Projects, Spring 2017 at Indiana University Bloomington.

REFERENCES

[1] "Kinesis - real-time streaming data in the aws cloud," Web Page, accessed: 2017-01-17. [Online]. Available: https://aws.amazon.com/kinesis/
[2] "Amazon Kinesis streams key concepts," Web Page, accessed: 2017-03-15. [Online]. Available: http://docs.aws.amazon.com/streams/latest/dev/key-concepts.html
[3] "Kinesis - real-time streaming data in the aws cloud," Web Page, accessed: 2017-03-15. [Online]. Available: http://docs.aws.amazon.com/streams/latest/dev/amazon-kinesis-streams.html
[4] "Kinesis - aws sdk for java," Web Page, accessed: 2017-03-15. [Online]. Available: https://aws.amazon.com/sdk-for-java/
[5] J. Varia and S. Mathew, "Overview of amazon web services," *Amazon Web Services*, pp. 1–22, Jan. 2014. [Online]. Available: http://w.emacromall.com/techpapers/Overview%20of%20Amazon%20Web%20Services.pdf
[6] P. Deyhim, "Kafka vs. kinesis," Datapipe, Inc., techreport, 2016, last accessed: 2017-03-17. [Online]. Available: http://go.datapipe.com/whitepaper-kafka-vs-kinesis.pdf

Robot Operating System (ROS): A Useful Overview

MATTHEW LAWSON[1]

[1]School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*]Corresponding authors: laszewski@gmail.com

paper2, March 25, 2017

The Open Source Robotics Foundation (OSRF) oversees the maintenance and development of the Robot Operating System (ROS). ROS provides an open-source, extensible framework upon which roboticists can build simple or highly complex operating programs for robots. Features to highlight include: a) ROS' well-developed, standardized intra-robot communication system; b) its sufficiently-large set of programming tools; c) its C++ and Python APIs; and, d) its extensive library of third-party packages to address a large proportion of roboticists software needs. The OSRF distributes ROS under the BSD-3 license.

Keywords: Cloud, I524, robot, ros, ROS

https://github.com/eunosm3/classes/blob/master/docs/source/format/report/report.pdf

1. ROBOT OPERATING SYSTEM (ROS)

1. Introduction The Open Source Robotics Foundation's middleware product *Robot Operating System*, or ROS, provides a framework for writing operating systems for robots. ROS offers "a collection of tools, libraries, and conventions [meant to] simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms" [?]. The Open Source Robotics Foundation, hereinafter OSRF or the Foundation, attempts to meet the aforementioned objective by implementing ROS as a modular system. That is, ROS offers a core set of features, such as inter-process communication, that work with or without pre-existing, self-contained components for other tasks.

2. Architecture The OSRF designed ROS as a distributed, modular system. The OSRF maintains a subset of essential features for ROS, i.e., the core functions upon which higher-level packages build, to provide an extensible platform for other roboticists. The Foundation also coordinates the maintenance and distribution of a vast array of ROS add-ons, referred to as modules. Figure 1 illustrates the ROS universe in three parts: a) the plumbing, ROS' communications infrastructure; b) the tools, such as ROS' visualization capabilities or its hardware drivers; and c) ROS' ecosystem, which represents ROS' core developers and maintainers, its contributors and its user base.

The modules or packages, which are analogous to packages in Linux repositories or libraries in other software distributions such as *R*, provide solutions for numerous robot-related challenges. General categories include a) drivers, such as sensor and actuator interfaces; b) platforms, for steering and image processing, etc.; c) algorithms, for task planning and obstacle avoidance; and, d) user interfaces, such as tele-operation and sensor data display. [?]

Communications Infrastructure | | General OSRF maintains three distinct communication methods for ROS: a) *message passing*; b) *services*; and, c) *actions*. Each method utilizes ROS' standard communication type, the *message* [?]. Messages, in turn, adhere to ROS' *interface description language*, or IDL. The IDL dictates that messages should be in the form of a data structure comprised of typed fields [?]. Finally, *.msg* files store the structure of messages published by various nodes so that ROS' internal systems can generate source code automatically.

Communications Infrastructure | | Message Passing ROS implements a publish-subscribe anonymous message passing system for inter-process communication, hereinafter pubsub, as its most-basic solution for roboticists. A pubsub system consists of two complementary pieces: a) a device, node or process, hereinafter node, publishing messages, i.e., information, to a *topic*; and b) another node *listening to* and ingesting the information from the associated topic. Pubsub's method of operation analogizes to terrestrial radio. In the analogy, the radio station represents the publishing node, the radio receiver maps to the subscribing node and the frequency on which one transmits and the other receives represents the topic.

The OSRF touts the pubsub communications paradigm as the ideal method primarily due to its anonymity and its requirement to communicate using its message format. With respect to the first point, the nodes involved in bilateral or multilateral conversations need only know the topic on which to publish or subscribe in order to communicate. As a result, nodes can be replaced, substituted or upgraded without changing a single line of code or reconfiguring the software in any manner. The subscriber node can even be deleted entirely without affecting images/ros-is.png

Fig. 1. A Conceptualization of What ROS, the *R*obot *O*perating *S*ystem, Offers to Roboticists [**?**]

any aspect of the robot except those nodes that depend on the deleted node.

In addition, ROS' pubsub requires well-defined interfaces between nodes in order to succeed. For instance, if a node publishes a message without a crucial piece information a subscribing node requires or in an unexpected format, the message would be useless. Alternatively, it would be pointless for an audio processing node to subscribe to a node publishing lidar data. Therefore, a message's structure must be well-defined and available for reference as needed in order to ensure compatibility between publisher and subscriber nodes. As a result, ROS has a modular communication system. That is, a subscriber node may use all or only parts of a publishing node's message. Further, the subscribing node can combine the data with information from another node before publishing the combined information to a different topic altogether for a third node's use. At the same time, a fourth and fifth node could subscribe to the original topic for each node's respective purpose.

Finally, ROS' pubsub can natively replay messages by saving them as files. Since a subscriber node processes messages received irrespective of the message's source, publishing a saved message from a subscriber node at a later time works just as well as an actual topic feed. One use of asynchronous messaging: postmortem analysis and debugging.

Communications Infrastructure | | Services ROS also provides a synchronous, real-time communication tool under the moniker *services* [**?**]. Services allow a subscribing node to request information from a publishing node instead of passively receiving whatever the publishing node broadcasts whenever it broadcasts it. A service consists of two messages, the request and the reply. It otherwise mirrors ROS' message passing function. Finally, users can establish a continuous connection between nodes at the expense of service provider flexibility.

Communications Infrastructure | | Actions ROS *actions* offer a more-advanced communication paradigm than either message passing or services [**?**]. Actions, which use the basic message structure from message passing, allow roboticists to create a request to accomplish some task, receive progress reports about the task completion process, receive task completion notifications and / or cancel the task request. For example, the roboticist may create a task, or equivalently, initiate an action, for the robot to conduct a laser scan of the area. The request would include the scan parameters, such as minimum scan angle, maximum scan angle and scan speed. During the process, the node conducting the scan will regularly report back its progress, perhaps as a value representing the percent of the scan completed, before returning the results of the scan, which should be a point cloud.

Tools | | Standard Messages ROS' extensive use in the robotics realm has allowed it to create message standards for various robot components [**?**]. Standard message definitions exist "for geometric concepts like poses, transforms, and vectors; for sensors like cameras, IMUs and lasers; and for navigation data like odometry, paths, and maps; among many others." These standards facilitate interoperability amongst robot components as well as easing development efforts by roboticists.

Tools | | Robot Geometry Library Robots with independently movable components, such as appendages (with joints) or movable sensors, must be able to coordinate such movements in order to be usable. Maintaining an accurate record of where a movable component is in relation to the rest of the robot presents a significant challenge in robotics [**?**].

`images/robotgeometry.png`

Fig. 2. A Simulated Robot with Many Coordinate Frames [**?**]

ROS addresses this issue with its *transform* library. The *tf* library tracks components of a robot using three-dimensional coordinate frames [**?**]. It records the relationship between coordinate frame positional values at sequential points in time in a tree structure. tf's built-in functions allow the roboticist to transform a particular coordinate frame's values to same basis as a different coordinate frame's values. As a result, the user, or the user's program, can always calculate any coordinate frame's relative position to any or all of the other coordinate frame positions at any point in time. Although the first-generation library, *tf*, has been deprecated in favor of the second-generation one, *tf2*, the Foundation and ROS users still refer to the library as tf.

Tools | | Robot Description Language ROS describes robots in a machine-readable format using its *Unified Robot Description Format*, or URDF [**?**]. The file delineates the physical properties of the robot in XML format. URDF files enable use of the tf library, useful visualizations of the robot and the use of the robot in simulations.

Tools | | Diagnostics ROS' diagnostics meta-package, i.e., a package of related packages, "contains tools for collecting, publishing, analyzing and viewing diagnostics data [**?**]." ROS' diagnostics take advantage of the aforementioned message system to allow nodes to publish diagnostic information to the standard diagnostic topic. The nodes use the diagnostic_updater and self_test packages to publish diagnostic information, while users can access the information using the rqt_robot_monitor package. ROS does not require nodes to include certain information in their respective publications, but diagnostic publications generally provide some standard, basic information. That information may include serial numbers, software versions, unique incident IDs, etc.

Tools | | Command Line Interfaces (CLI) ROS provides at least 45 command line tools to the roboticist [?]. Therefore, ROS can be setup and run entirely from the command line. However, the GUI interfaces remain more popular among the user-base. Examples of ROS CLI tools include: a) *rosmsg*, which allows the user to examine messages, including the data structure of .msg files [?]; b) *rosbag*, a tool to perform various operations on .bag files, i.e., saved node publications; and, c) *rosbash*, which extends bash, a Linux shell program, with ROS-related commands.

Tools | | Graphical User Interfaces (GUI) OSRF includes two commonly-used GUIs, *rviz* and *rqt* [?], in the core ROS distribution. rviz creates 3D visualizations of the robot, as well as the sensors and sensor data specified by the user. This component renders the robot in 3D based on a user-supplied URDF document. If the end-user wants or needs a different GUI, s/he can use rqt, a Qt-based GUI development framework. It offers plug-ins for items such as: a) viewing layouts, like tabbed or split-screens; b) network graphing capabilities to visualize the robot's nodes; c) charting capabilities for numeric values; d) data logging displays; and, e) topic (communication) monitoring.

Ecosystem ROS benefits from a wide-ranging network of interested parties, including core developers, package contributors, hobbyists, researchers and for-profit ventures. Although quantifiable use metrics for ROS remain scarce, ROS does have more than 3,000 software packages available from its community of users [?], ranging from proof-of-concept algorithms to industrial-quality software drivers. Corporate users include large organizations such as Bosch (Robert Bosch GmbH) and BMW AG, as well as smaller companies such as ClearPath Robotics, Inc. and Stanley Innovation. University users include the Georgia Institute of Technology and the University of Arizona, among others [?].

3. API ROS supports robust application program interfaces, APIs, through libraries for C++ and Python. It provides more-limited, and experimental, support for nodejs, Haskell and Mono / .NET programming languages, among others. The latter library opens up use with C# and Iron Python [?].

4. Licensing The OSRF distributes the core of ROS under the standard, three-clause BSD license, hereinafter BSD-3 license. The BSD-3 license belongs to a broader class of copyright licenses referred to as *permissive licenses* because it imposes zero restrictions on the software's redistribution as long as the redistribution maintains the license's copyright notices and warranty disclaimers [?].

Other names for BSD-3 include: a) BSD-new; b) New BSD; c) revised BSD; d) The BSD License, the official name used by the Open Source Initiative; and, e) Modified BSD License, used by the Free Software Foundation.

Although the OSRF distributes the main ROS elements under the BSD-3 license, it does not require package contributors or end-users to adopt the same license. As a result, full-fledged ROS programs may include other types of *Free and Open-Source Software* [?], or FOSS, licenses. In addition, programs may depend on proprietary or unpublished drivers unavailable to the broader community.

5. Use Cases ROS' end-markets, its use cases, include manipulator robots, i.e., robotic arms with grasping units; mobile robots, such as autonomous, mobile platforms; autonomous cars; social robots; humanoid robots, unmanned / autonomous vehicles; and an assortment of other robots [?].

5.1. Use Cases for Big Data Fetch Robotics, Inc. offers its *Automated Data Collection Platform* robot to the market so corporations can "[g]ather environmental data more frequently and more accurately for [its] Internet of Things...and Big Data Applications. [?]" Fetch's system automatically collects data, such as RFID tracking (inventory management) or in-store shelf surveys. The latter service began in January 2017 when Fetch partnered with Trax Image Recognition, which makes image recognition software [?].

6. Educational material Those interested in learning more about OSRF's ROS should visit ROS' homepage, www.ros.org or its wiki page, wiki.ros.org In addition, ClearPath Robotics maintains a useful set of tutorials at https://goo.gl/hRmM3k. Finally, several books dedicated to programming with ROS, such as *Programming Robots with ROS: A Practical Introduction to the Robot Operating System* by Quibley, Gerkey and Smart can be purchased at retail.

7. Conclusion ROS offers a number of attractive features to its users, including a well-developed and standardized intra-robot communication system, modular design, vetted third-party additions and legitimacy via real-world applications. Although its status as open source software precludes direct support from its parent organization, OSRF, for-profit organizations and the software's active community of users provide reassurances to any roboticist worried about encountering a seemingly-insurmountable problem.

Acknowledgment I would like to thank my employer, Indiana Farm Bureau, for its support of my continuing education.

AUTHOR BIOGRAPHIES

Matthew Lawson received his BSBA, Finance in 1999 from the University of Tennessee, Knoxville. His research interests include data analysis, visualization and behavioral finance.

A. WORK BREAKDOWN

The work on this project was distributed as follows between the authors:

Matthew Lawson. Matthew researched and wrote all of the material for this paper.

Facebook Tao

NANDITA SATHE[1,*]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding author: nsathe@iu.edu

paper-2, March 25, 2017

As early as 2005, Facebook started using MySQL, a relational database coupled with large distributed cache called memcache. Facebook soon realized however efficient relational database it would use, it is not sufficient to manage the enormous data challenge Facebook had. The data was a social graph. Another mismatch, which relational database or block cache had was, most of the data that would be read into cache did not belong to any relation. For example, 'If user likes that picture". In most records the answer would be 'No' or 'False'. Storing and reading this unwanted data was a burden. Meanwhile Facebook users' base was increasing daily. Ultimately Facebook came up with Facebook Tao, a distributed social graph data store.

Keywords: Facebook Tao, Graph Database, memcache

https://github.com/nsathe/sp17-i524/blob/master/paper2/S17-IO-3017/report.pdf

1. INTRODUCTION

In the paper published in USENIX annual technical conference, Facebook Inc describes TAO (The Association and Objects) as [1] a geographically distributed data store that provides timely access to the social graph for Facebook's demanding workload using a fixed set of queries. It is deployed at Facebook for many data types that fit its model. The system runs on thousands of machines, is widely distributed, and provides access to many petabytes of data. TAO represents social data items as Objects (user) and relationship between them as Associations (liked by, friend of). TAO cleanly separates the caching tiers from the persistent data store allowing each of them to be scaled independently. To any user of the system it presents a single unified API that makes the entire system appear like 1 giant graph database [2]. Key advantages of the system include [2]:

- Provides a clean separation of application/product logic from data access by providing a simple yet powerful graph API and data model to store and fetch data. This enables Facebook product engineers to move fast.
- By implementing a write-through cache TAO allows Facebook to provide a better user experience and preserve the all important read-what-you-write consistency semantics even when the architecture spans multiple geographical regions.
- By implementing a read-through write-through cache TAO also protects the underlying persistent stores better by avoiding issues like thundering herds without compromising data consistency.

2. TAO'S GOAL

Main goal of implementing Tao is efficiently scaling the data. Facebook handles approximately a billion requests per second. So obviously data store has to be scalable. More than that, scalability should be efficient otherwise scaling data across machines would be extremely costly.

Second goal is to achieve lowest possible read latency. So that if a user has commented on a post, the original post writer should be able to read it immediately. Efficiency in Scaling and low Read latency is achieved by (i) separating cache and data storage, (ii) Graph specific caching and (iii) Sub-dividing data centers [3].

Third goal is to achieve timeliness of writes. If a web server has written something and it sends a read request, it should be able to read the post. Write timeliness is achieved by (i) Write trough cache and (ii) Asynchronous replication [3].

Lastly high read availability for the same reasons mentioned above, which is achieved by using alternate data sources.

3. TAO DATA MODEL AND API

Facebook Inc. explains TAO data model and API associated with using a simple example [4]. Figure 1 depicts TAO data model.

This simple example shows a subgraph of objects and associations that is created in TAO after Alice checks in at the Golden Gate Bridge and tags Bob there, while Cathy comments on the check-in and David likes it. Every data item, such as a user, check-in, or comment, is represented by a typed object containing a dictionary of named fields. Relationships between objects, such as “liked by" or “friend of," are represented by typed edges

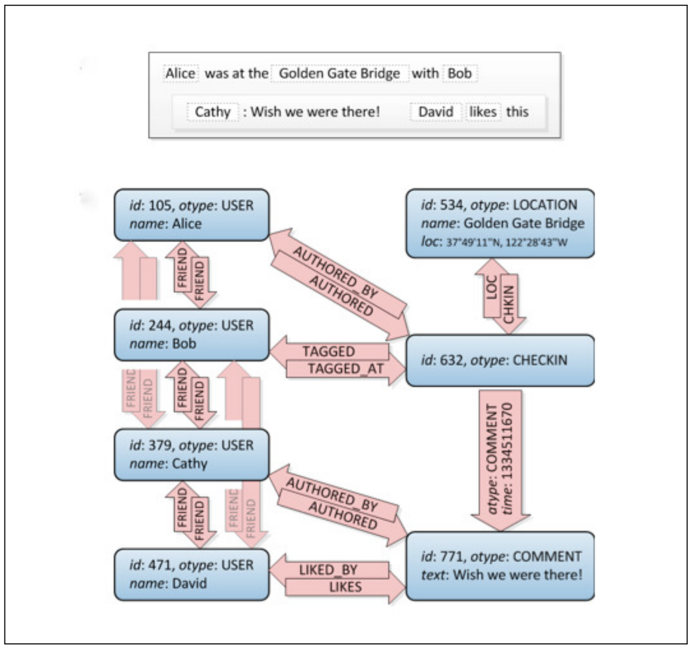


Fig. 1. TAO Data Model [4].

(associations) grouped in association lists by their origin. Multiple associations may connect the same pair of objects as long as the types of all those associations are distinct. Together objects and associations form a labeled directed multigraph.

For every association type a so-called inverse type can be specified. Whenever an edge of the direct type is created or deleted between objects with unique IDs id1 and id2, TAO will automatically create or delete an edge of the corresponding inverse type in the opposite direction (id2 to id1). The intent is to help the application programmer maintain referential integrity for relationships that are naturally mutual, like friendship, or where support for graph traversal in both directions is performance critical, as for example in “likes” and “liked by."

3.1. Objects and Associations

[1] TAO objects are typed nodes, and TAO associations are typed directed edges between objects. Objects are identified by a 64-bit integer (id) that is unique across all objects, regardless of object type (otype). Associations are identified by the source object (id1), association type (atype) and destination object (id2). At most one association of a given type can exist between any two objects. Both objects and associations may contain data as key $->$ value pairs. A per-type schema lists the possible keys, the value type, and a default value. Each association has a 32-bit time field, which plays a central role in queries.

Object: (id) -> (otype, (key -> value)*)

Assoc.: (id1, atype, id2) -> (time, (key -> value)*)

Figure 1 shows how TAO objects and associations might encode the example, with some data and times omitted for clarity. The example’s users are represented by objects, as are the checkin, the landmark, and Cathy’s comment. Associations capture the users’ friendships, authorship of the checkin and comment, and the binding between the checkin and its location and comments.

The set of operations on objects is of the fairly common create/set-fields/get/delete variety. All objects of a given type have the same set of fields. New fields can be registered for an object type at any time and existing fields can be marked deprecated by editing that type’s schema. In most cases product engineers can change the schema of their types without any operational work.

Associations are created and deleted as individual edges. If the association type has an inverse type defined, an inverse edge is created automatically. The API helps the data store exploit the creation-time locality of workload by requiring every association to have a special time attribute that is commonly used to represent the creation time of association. TAO uses the association time value to optimize the working set in cache and to improve hit rate.

There are three main classes of read operations on associations [4]:

- Point queries look up specific associations identified by their (id1, type, id2) triplets. Most often they are used to check if two objects are connected by an association or not, or to fetch data for an association.

- Range queries find outgoing associations given an (id1, type) pair. Associations are ordered by time, so these queries are commonly used to answer questions like "What are the 50 most recent comments on this piece of content?" Cursor-based iteration is provided as well.

- Count queries give the total number of outgoing associations for an (id1, type) pair. TAO optionally keeps track of counts as association lists grow and shrink, and can report them in constant time

4. TAO ARCHITECTURE

This section describes the architecture of TAO. TAO is separated into layers: two caching layers and a storage layer.

4.1. Storage Layer

The data is persisted using MySQL. The API is mapped to a small number of SQL queries. Data is divided into logical shards. By default all object types are stored in one table and association in others. Every ’object-id’ has a corresponding ’shard-id’. Objects are bounded to a single shard throughout their lifetime. An association is stored on the shard of its id1, so that every association query can be served from a single server [3].

4.2. Caching Layer

TAO’s cache implements the complete API for clients, handling all communication with databases. A region/tier is made of multiple closely located Data centers. Multiple Cache Serves make up a tier (set of databases in a region are also called a tier) that can collectively capable of answering any TAO Request. Each cache request maps to a server based on sharding. The cache is filled based on a LRU policy. Write operations on an association with an inverse may involve two shards, since the forward edge is stored on the shard for id1 and the inverse edge is on the shard for id2. Handling writes with multiple shards involve: Issuing an RPC call to the member hosting id2, which will contact the database to create the inverse association. Once the inverse write is complete, the caching server issues a write to the database for id1. TAO does not provide atomicity between the two updates. If a failure occurs the forward may exist without an inverse, these hanging associations are scheduled for repair by an asynchronous job [3].

4.3. Leaders and Followers

There are two tiers of caching clusters in each geographical region. Clients talk to the first tier, called followers. If a cache miss occurs on the follower, the follower attempts to fill its cache from a second tier, called a leader. Leaders talk directly to a MySQL cluster in that region. All TAO writes go through followers to leaders. Caches are updated as the reply to a successful write propagates back down the chain of clusters. Leaders are responsible for maintaining cache consistency within a region. They also act as secondary caches, with an option to cache objects and associations in Flash [4].

4.4. Scaling Geographically

High read workload scales with total number of follower servers. The assumption is that latency between followers and leaders is low. Followers behave identically in all regions, forwarding read misses and writes to the local region's leader tier. Leaders query the local region's database regardless of whether it is the master or slave. This means that read latency is independent of inter-region latency. Writes are forwarded by the local leader to the leader that is in the region with the master database. Read misses by followers are 25X as frequent as writes in the workload thus read misses are served locally. Facebook chooses data center locations that are clustered into only a few regions, where the intra-region latency is small (typically less than 1 millisecond). It is then sufficient to store one complete copy of the social graph per region.

Since each cache hosts multiple shards, a server may be both a master and a slave at the same time. It is preferred to locate all of the master databases in a single region. When an inverse association is mastered in a different region, TAO must traverse an extra inter-region link to forward the inverse write. TAO embeds invalidation and refill messages in the database replication stream. These messages are delivered in a region immediately after a transaction has been replicated to a slave database. Delivering such messages earlier would create cache inconsistencies, as reading from the local database would provide stale data. If a forwarded write is successful then the local leader will update its cache with the fresh value, even though the local slave database probably has not yet been updated by the asynchronous replication stream. In this case followers will receive two invalidates or refills from the write, one that is sent when the write succeeds and one that is sent when the write's transaction is replicated to the local slave database [3].

5. EDUCATIONAL MATERIAL

To get started on learning Facebook TAO, following resources can prove helpful.

- [1] - Technical paper on Facebook TAO.
- [4] - Background, Architecture and Implementation from Facebook itself.
- [5] - TAO summary in a video on USENIX website.

6. ACKNOWLEDGEMENTS

The author would like to thank Prof. Gregor von Laszewski and his associates from the School of Informatics and Computing for providing all the technical support and assistance.

REFERENCES

[1] N. Bronson *et al.*, "Tao: Facebook's distributed data store for the social graph," in *2013 USENIX Annual Technical Conference*, 2013. [Online]. Available: http://ai2-s2-pdfs.s3.amazonaws.com/39ac/2e0fc4ec63753306f99e71e0f38133e58ead.pdf

[2] V. Venkataramani, "What is the tao cache used for at facebook," Web Page, June 2013. [Online]. Available: https://www.quora.com/What-is-the-TAO-cache-used-for-at-Facebook

[3] N. Upreti, "Facebook's tao and unicorn data storage and search platforms," Slides, April 2015. [Online]. Available: https://www.slideshare.net/nitishupreti/faceboko-tao-unicorn

[4] M. Marchukov, "Tao: The power of the graph," Web Page, June 2013. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/tao-the-power-of-the-graph/10151525983993920/

[5] N. Bronson, "Tao: Facebook's distributed data store for the social graph," Slides, June 2013. [Online]. Available: https://www.usenix.org/node/174510

Retainable Evaluator Execution Framework

PRATIK JAIN

School of Informatics and Computing, Bloomington, IN 47408, U.S.A.

Corresponding authors: jainps@iu.edu

Paper-2, March 25, 2017

Apache REEF is a Big Data system that makes it easy to implement scalable, fault-tolerant runtime environments for a range of data processing models on top of resource managers such as Apache YARN and Mesos. The key features and abstractions of REEF are discussed. Two libraries of independent value are introduced. Wake is an event-based-programming framework and Tang is a dependency injection framework designed specifically for configuring distributed systems.

Keywords: REEF, Tang, Wake

https://github.com/pratik11jain/sp17-i524/blob/master/paper2/S17-IR-2012/report.pdf

1. INTRODUCTION

With the continuous growth of Hadoop the range of computational primitives expected by its users has also broadened. A number of performance and workload studies have shown that Hadoop MapReduce is a poor fit for iterative computations, such as machine learning and graph processing. Also, for extremely small computations like ad-hoc queries that compose the vast majority of jobs on production clusters, Hadoop MapReduce is not a good fit. Hadoop 2 addresses this problem by factoring MapReduce into two components: an application master that schedules computations for a single job at a time, and YARN, a cluster resource manager that coordinates between multiple jobs and tenants. In spite of the fact that this resource manager, YARN, allows a wide range of computational frameworks to coexist in one cluster, many challenges remain [1].

From the perspective of the scheduler, a number of issues arise that must be appropriately handled in order to scale-out to massive datasets. First, each map task should be scheduled close to where the input block resides, ideally on the same machine or rack. Second, failures can occur at the task level at any step, requiring backup tasks to be scheduled or the job being aborted. Third, performance bottlenecks can cause an imbalance in the task-level progress. The scheduler must react to these stragglers by scheduling clones and incorporating the logical task that crosses the finish line first.

Apache REEF (Retainable Evaluator Execution Framework), a library for developing portable applications for cluster resource managers such as Apache Hadoop YARN or Apache Mesos, addresses these challenges. It provides a reusable control-plane for scheduling and coordinating task-level work on cluster resource managers. The REEF design enables sophisticated optimizations, such as container re-use and data caching, and facilitates workflows that span multiple frameworks. Examples include pipelining data between different operators in a relational system, retaining state across iterations in iterative or recursive data flow, and passing the result of a MapReduce job to a Machine Learning computation.

2. FEATURES

Due to its following critical features, Apache REEF drastically simplifies development of resource managers [2].

2.1. Centralized Control Flow

Apache REEF turns the chaos of a distributed application into various events in a single machine. These events include container allocation, Task launch, completion, and failure.

2.2. Task runtime

Apache REEF provides a Task runtime which is instantiated in every container of a REEF application and can keep data in memory in between Tasks. This enables efficient pipelines on REEF.

2.3. Support for multiple resource managers

Apache REEF applications are portable to any supported resource manager with minimal effort. In addition to this, new resource managers are easy to support in REEF.

2.4. .NET and Java API

Apache REEF is the only API to write YARN or Mesos applications in .NET. Additionally, a single REEF application is free to mix and match tasks written for .NET or Java.

2.5. Plugins

Apache REEF allows for plugins to augment its feature set without hindering the core. REEF includes many plugins, such as a name-based communications between Tasks, MPI-inspired group communications, and data ingress.

As a result of such features Apache REEF shows properties like retainability of hardware resources across tasks and jobs, composability of operators written for multiple computational frameworks and storage backends, cost modeling for data movement and single machine parallelism, fault handling [3] and elasticity.

3. KEY ABSTRACTIONS

REEF is structured around the following key abstractions [4]:

Driver: This is a user-supplied control logic that implements the resource allocation and Task scheduling logic. There is exactly one Driver for each Job. The duration and characteristics of the Job are determined by this module.

Task: This encapsulates the task-level client code to be executed in an Evaluator.

Evaluator: This is a runtime environment on a container that can retain state within Contexts and execute Tasks (one at a time). A single evaluator may run many activities throughout its lifetime. This enables sharing among Activities and reduces scheduling costs.

Context: It is a state management environment within an Evaluator that is accessible to any Task hosted on that Evaluator.

Services: Objects and daemon threads that are retained across Tasks that run within an Evaluator [1]. Examples include caches of parsed data, intermediate state, and network connection pools.

4. WAKE AND TANG

The lower levels of REEF can be decoupled from the data models and semantics of systems built atop it. This results in two standalone systems, Tang and Wake which are both language independent and allow REEF to bridge the JVM and .NET.

Tang is a configuration management and checking framework [5]. It emphasizes explicit documentation and automatic checkability of configurations and applications instead of ad-hoc, application-specific configuration and bootstrapping logic. It not only supports distributed, multi-language applications but also gracefully handles simpler use cases. It makes use of dependency injection to automatically instantiate applications. Given a request for some type of object, and information that explains how dependencies between objects should be resolved, dependency injectors automatically instantiate the requested object and all of the objects it depends upon. Tang makes use of a few simple wire formats to support remote and even cross-language dependency injection.

Wake is an event-driven framework based on ideas from SEDA, Click, Akka and Rx [6]. It is general purpose in the sense that it is designed to support computationally intensive applications as well as high-performance networking, storage, and legacy I/O systems. Wake is implemented to support high-performance, scalable analytical processing systems i.e. big data applications. It can be used to achieve high fanout and low latency as well as high-throughput processing and it can thus aid to implement control plane logic and the data plane.

Wake is designed to work with Tang. This makes it extremely easy to wire up complicated graphs of event handling logic. In addition to making it easy to build up event-driven applications, Tang provides a range of static analysis tools and provides a simple aspect-style programming facility that supports Wake's latency and throughput profilers.

5. RELATIONSHIPS WITH OTHER APACHE PRODUCTS

Given REEF's position in the big data stack, there are three relationships to consider: Projects that fit below, on top of, or alongside REEF in the stack [**?**].

5.1. Below REEF

REEF is designed to facilitate application development on top of resource managers like Mesos and YARN. Hence, its relationship with the resource managers is symbiotic by design.

5.2. On Top of REEF

Apache Spark, Giraph, MapReduce and Flink are some of the projects that logically belong at a higher layer of the big data stack than REEF. Each of these had to individually solve some of the issues REEF addresses.

5.3. Alongside REEF

Apache builds library layers on top of a resource management platform. Twill, Slider, and Tez are notable examples in the incubator. These projects share many objectives with REEF. Twill simplifies programming by exposing a programming model. Apache Slider is a framework to make it easy to deploy and manage long-running static applications in a YARN cluster. Apache Tez is a project to develop a generic Directed Acyclic Graph (DAG) processing framework with a reusable set of data processing primitives. Apache Helix automates application-wide management operations which require global knowledge and coordination, such as repartitioning of resources and scheduling of maintenance tasks. Helix separates global coordination concerns from the functional tasks of the application with a state machine abstraction.

6. CONCLUSION

REEF is a flexible framework for developing distributed applications on resource manager services. It is a standard library of reusable system components that can be easily composed into application logic. It possesses properties of retainability, composability, cost modeling, fault handling and elasticity. Its key components are driver, task, evaluator, context and services. Its relationship with projects above it, below it and alongside it in the big data stack is discussed. Thus, a brief overview of REEF is shown here.

ACKNOWLEDGEMENTS

The author thanks Prof. Gregor von Laszewski and all the course AIs for their continuous technical support.

REFERENCES

[1] Byung-Gon Chun, Chris Douglas, Shravan Narayanamurthy, Josh Rosen, Tyson Condie, Sergiy Matusevych, Raghu Ramakrishnan, Russell Sears, Carlo Curino, Brandon Myers, Sriram Rao, Markus

Weimer, “Reef: Retainable evaluator execution framework,” in *VLDB Endowment, Vol. 6, No. 12*, Aug. 2013. [Online]. Available: http://db.disi.unitn.eu/pages/VLDBProgram/pdf/demo/p841-sears.pdf

[2] The Apache Software Foundation, “Apache reef™ - a stdlib for big data,” Web Page, Nov. 2016, accessed 2017-03-15. [Online]. Available: http://reef.apache.org/

[3] Techopedia Inc., “Retainable evaluator execution framework (reef),” Web Page, Jan. 2017, accessed 2017-03-17. [Online]. Available: https://www.techopedia.com/definition/29891/retainable-evaluator-execution-framework-reef

[4] Markus Weimer, Yingda Chen, Byung-Gon Chun, Tyson Condie, Carlo Curino, Chris Douglas, Yunseong Lee, Tony Majestro, Dahlia Malkhi, Sergiy Matusevych, Brandon Myers, Shravan Narayanamurthy, Raghu Ramakrishnan, Sriram Rao, Russell Sears, Beysim Sezgin, Julia Wang, “Reef: Retainable evaluator execution framework,” in *Proc ACM SIGMOD Int Conf Manag Data. Author manuscript*, Jan. 2016, pp. 1343–1355. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4724804/

[5] The Apache Software Foundation, “Tang,” Web Page, Nov. 2016, accessed 2017-03-15. [Online]. Available: http://reef.apache.org/tang.html

[6] ——, “Wake,” Web Page, Dec. 2016, accessed 2017-03-15. [Online]. Available: http://reef.apache.org/wake.html

OpenStack Nova: Compute Service of OpenStack Cloud

KUMAR SATYAM[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: ksatyam@indiana.edu

paper-2, March 25, 2017

OpenStack Nova is the compute service of the OpenStack cloud system.It is designed to manage and automate the pools of computer resources and can work on bare metal and high performance computing. It is written in python.We will discuss the main components included in the Nova Architecture [1].

Keywords: Cloud,OpenStack,Nova, API, I524

https://github.com/satyamsah/sp17-i524/blob/master/paper2/S17-IR-2031/report.pdf

INTRODUCTION

Nova is responsible for spawning vm instances in openstack environment. It is built on a messaging architecture which runs on several servers. This architecture allows components to communicate through a messaging queue. Nova together shares a centralized SQL-based database for smaller deployment. For large deployments an aggregation is used to manage data across multiple data stores[2].

NOVA FRAMEWORK

Nova is comprised of multiple server processes, each performing different functions. The OpenStack provided user interface for Nova which is a REST API. During invocation of the API, the Nova communicates via RPC (Remote procedure call) passing mechanism.

The API servers process REST requests, which typically involve database reads/writes. RPC messaging is done via the 'oslo.message' library. Most of the nova components can run on different servers and have a manager that is listening for RPC messages. One of the components is Nova Compute where a single process runs on the hypervisor it is managing.

Nova has a centralized database that is logically shared between all components[4].

NOVA COMPONENTS

Below are the major components of Nova:

DB: An SQL database for data storage.This is the SQLAlchemy-compatible database. The database name is 'nova'. The 'nova-conductor' service is the only service that writes to the database.

ARCHITECTURE -OPENSTACK NOVA

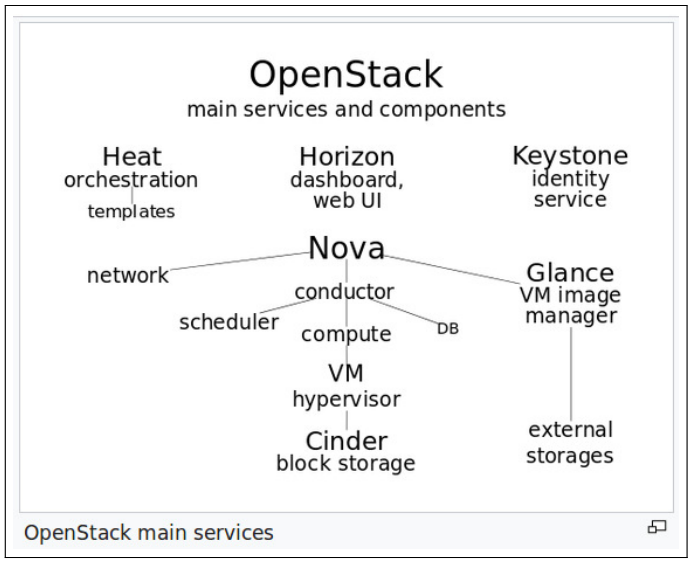


Fig. 1. Architecture of OpenStack cloud using Nova [3]

API: It is the component that receive HTTP requests, converts commands and communicates with other components via the 'oslo.messeging' queue or HTTP. The 'python-novaclient 7.1.0' is a python binding to OpenStack Nova API which acts as a client for the same. The nova-api provides an endpoint for all API queries (either OpenStack API or EC2 API), initiates most of the orchestration activities (such as running an instance) and also enforces some policy (mostly quota checks)

Schedular: The 'nova-schedular' is a service to determine how to dispatch compute requests.For example, it determines on which host a VM should launch. Compute is configured with a default scheduler options in the /etc/nova/nova.conf file[5].

Network: It manages IP forwarding, bridges, and vlans. The network controller with nova-network provides virtual networks to enable compute servers to interact with each other and with public network. Compute with nova-network support the following modes, which are implemented as Network Manager types:

- Flat Network Manager
- Flat DHCP Network Manager
- VLAN Network Manger

volume : It manages creation, attaching and detaching of persistent volumes to compute instances. This functionality is being migrated to Cinder, a separate OpenStack service

Compute: It manages communication with hypervisor and virtual machine

Conductor: It handles requests that need coordination, acts as a database proxy, or handles object conversions

STEP BY STEP INVOCATION OF OPENSTACK NOVA SERVICE

- Authenticating against the nova-api endpoint
- Listing instances
- Retrieving an instance by name and shutting it down
- Booting an instance and checking status
- Attaching Floating IP address
- Changing a security group
- Retrieving console login

NOVA AND OTHER OPENSTACK SERVICE

Nova is the main Openstack services as it interact with all the services to set IAAS stack. Nova interact with Glance service to provided images for VM provisioning.It also interact with the Queue service via a nova-scheduler and nova-conductor API.It interacts with Keystone for authentication and authorization services. It also interacts with Horizon for web interface.

NOVA DEPENDENCE ON AMQP

AMQP is the messaging technology chosen by the OpenStack cloud.The AMQP sits between any two Nova components and allow them to communicate in a loosely coupled fashion. Nova Components use RPC to communicate to one another.It is build on the pub/sub paradigm to have the benefits. Decoupling between client and server(such that the client does not need to know where the servant's reference is) is a major advantage of AMQP[6].

ORCHESTRATION TASK IN NOVA

Nova-Conductor service plays an important role to manage the execution of workflows which involve the scheduler. Rebuild, resize, and building the instance are managed here. This was done in order to have better separation of responsibilities between what compute nodes should handle and what schedular should handle and to clean up the path of execution. In order to query the schedular in a synchronous manner it needed to happen after the API had returned a response otherwise API response times would increase and thats why conductor was chosen. And changing the schedular call from asynchronous to synchronous helped to clean up the code[7].

The earlier logic was complicated and the scheduling logic was distributed across the code.The earlier process was changed to the following:

- API receives request to build an instance.
- API sends an RPC cast to conductor to build an instance.
- Conductor sends an RPC call to schedular to pick a compute and waits for the response. If there is a schedular fail, it stops the build at the conductor.
- Conductor sends an RPC cast to the compute to build the instance. If the build succeeds, stop here. If it fails then the compute sends an RPC cast to conductor to build an instance. This is the same RPC message that was sent by the API.

OPENSTACK NOVA SUPPORT

Earlier Openstack was supported on KVM but its support has been extended QEMU. Microsoft Hyper -V and Vmware ESXi too provide extended support. Nova has support for XenServer and XCP through XenAPI virtual layer.It also support bare metal deployment and provisioning from 'Ironic' version. This means it is possible to deploy virtual machines. By default, it will use PXE and IPMI to provision and turn on/off the machine. But from the Ironic version it support vendor-specific plugins which may implement additional functionality.

OPENSTACK NOVA IN BIGDATA

A use case has been developed to leverage OpenStack to perform big data analytics. OpenStack used cassandra database for columnar data structure. PostgreSQL for relational data structure.This use case also allows us to configure Hadoop distributed file system for large unstructured data. OpenStack Sahara has come up with core cloud components of Big data[8].

OPENSTACK NOVA AND OTHER COMPETITORS

The OpenStack nova does the same task as it is being done by AWS EC2, Google CE and Microsoft Azure VM. With respect to Beach marking the main difference is the cloud administrator can upload their images in OpenStack where as in AWS and Google Cloud storage, one need to use the pre-defined list.The AWS is used mainly as public cloud where as the Openstack can be used a private cloud. But OpenStack has an advantage of customizing our own cloud configuration which is not there in any of the vendor specific clouds.

CONCLUSION

We discussed about the main components of OpenStack Nova and how it is interacting with different other Openstack services.We also showed use case of running big data problem on Openstack. We also discussed the compute service offerings provided by cloud vendors other than OpenStack.

REFERENCES

[1] Wikipedia, “Openstack nova wikipedia,” accessed: 03-23-2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenStack

[2] OpenStack, “Openstack nova official,” accessed: 03-23-2017. [Online]. Available: https://docs.openstack.org/developer/nova/architecture.html

[3] Wikipedia, “Openstack nova bigdata,” accessed: 03-23-2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenStack

[4] P. Website, “Openstack nova on pepple website,” accessed: 03-23-2017. [Online]. Available: http://ken.pepple.info/openstack/2011/04/22/openstack-nova-architecture/

[5] O. Website, “Openstack nova schedular,” accessed: 03-23-2017. [Online]. Available: https://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html

[6] ——, “Openstack nova amqp,” accessed: 03-23-2017. [Online]. Available: https://docs.openstack.org/developer/nova/rpc.html

[7] ——, “Openstack nova orchestrator,” accessed: 03-23-2017. [Online]. Available: https://docs.openstack.org/developer/nova/conductor.html

[8] ——, “Openstack nova bigdata,” accessed: 03-23-2017. [Online]. Available: https://www.openstack.org/summit/san-diego-2012/openstack-summit-sessions/presentation/big-data-on-openstack-a-rackspace-use-case