

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación



**ANÁLISIS DE SINERGIAS EN DESPLIEGUE DE
SISTEMAS DE COMPUTACIÓN DISTRIBUIDA:
APACHE SPARK-OPENSTACK**

TRABAJO FIN DE MÁSTER

Pedro M. Verdugo Rodríguez

2015

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

**Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos**

TRABAJO FIN DE MÁSTER

**ANÁLISIS DE SINERGIAS EN DESPLIEGUE DE
SISTEMAS DE COMPUTACIÓN DISTRIBUIDA:
APACHE SPARK-OPENSTACK**

Autor

Pedro M. Verdugo Rodríguez

Director

Joaquín Salvachúa Rodríguez

Departamento de Ingeniería de Sistemas Telemáticos

2015

Resumen

El objetivo del siguiente trabajo es la búsqueda de posibles simplificaciones en el funcionamiento de sistemas orientados a clusterización de máquinas en entornos de computación distribuida.

Para ello analizaremos las infraestructuras de implementación más común, basadas en tecnologías libres de amplio y extendido uso: el sistema Apache Spark, encargado de la distribución de procesos en clústeres preexistentes, el sistema Hadoop, basado en la definición de procesos de trabajo para clústers, el proyecto OpenStack Sahara, basado en facilitar la gestión de clústeres Hadoop y finalmente la familia de aplicaciones OpenStack, encargada del despliegue de entornos operativos conjuntos sobre diversas máquinas físicas.

Haremos especial hincapié en el proyecto OpenStack Sahara, por su cercanía a nuestro campo de estudio.

Una vez familiarizados con el funcionamiento y despliegue de dichos sistemas, centraremos nuestro esfuerzo en la reducción de componentes superfluos o redundantes de nuestra arquitectura, para lo que realizaremos un despliegue de pruebas, garantizando mediante medidas cuantificables la optimización del entorno de ejecución de procesos de nuestro entorno.

Finalmente expondremos los resultados obtenidos, analizando objetivamente su relevancia así como coste de implementación.

Abstract

The goal of this paper is the research of possible simplifications in the operation of clustering oriented systems for distributed computing environments.

As such, we will analyze the most commonly implemented infrastructures, based on open source technologies with widespread use: Apache Spark, a system responsible for the distribution of processes in existing clusters, the Hadoop system based on the definition of job processes to clusters, the Sahara OpenStack project, based on facilitating Hadoop cluster management and finally the family of OpenStack applications, responsible for the deployment of several operating environments on different physical machines.

We have emphasized the Sahara OpenStack project, due to its proximity to our field of study.

Once familiar with the operation and deployment of such systems, we will focus our efforts on reducing unnecessary or redundant components in our architecture, for which we will proceed with a test deployment, using quantifiable measures to ensure a optimized process execution in our environment.

Finally we will present the obtained results, objectively analyzing their relevance and implementation cost.

Índice general

| | |
|------------------------------------|-----|
| Resumen..... | i |
| Abstract..... | iii |
| Índice general..... | v |
| Índice de figuras..... | vii |
| Siglas..... | ix |
| 1 Introducción..... | 10 |
| 1.1 Motivación..... | 10 |
| 1.2 Sistemas Libres..... | 10 |
| 1.3 Objetivos..... | 11 |
| 1.4 Estructura del Documento..... | 11 |
| 2 Estado del Arte..... | 13 |
| 2.1 Apache Spark[1]..... | 13 |
| 2.1.1 Características..... | 13 |
| 2.1.2 Arquitectura[2]..... | 13 |
| 2.1.3 Componentes[7]..... | 14 |
| 2.2 Apache Hadoop..... | 16 |
| 2.2.1 Características..... | 16 |
| 2.2.2 Arquitectura..... | 17 |
| 2.2.3 Componentes[8]..... | 19 |
| 2.3 OpenStack Sahara[9]..... | 21 |
| 2.3.1 Características[10]..... | 21 |
| 2.3.2 Arquitectura..... | 23 |
| 2.3.3 Componentes[12]..... | 25 |
| 2.3.4 Herramientas Opcionales..... | 28 |
| 2.4 Virtualización OpenSource..... | 30 |

| | | |
|-------|--|----|
| 2.4.1 | Virtualización Hardware..... | 30 |
| 2.4.2 | Virtualización de Sistema Operativo..... | 32 |
| 2.5 | OpenStack [19]..... | 33 |
| 2.5.1 | Características [20] | 33 |
| 2.5.2 | Arquitectura..... | 33 |
| 2.5.3 | Componentes [21] | 36 |
| 3 | Análisis de Funcionalidades..... | 38 |
| 3.1 | Usos Finales..... | 38 |
| 3.1.1 | Propósito General..... | 38 |
| 3.1.2 | Orientado a Computación | 38 |
| 3.1.3 | Orientado a Almacenamiento..... | 39 |
| 3.1.4 | Orientado a Red..... | 39 |
| 3.1.5 | Multi-Sitio | 39 |
| 3.1.6 | Híbrido..... | 40 |
| 3.1.7 | Escalable Masivo..... | 40 |
| 3.2 | Selección de Caso de Uso..... | 40 |
| 3.3 | Estructuras Propuestas..... | 40 |
| 3.3.1 | Estructura Clásica | 41 |
| 3.3.2 | Estructura Mínima..... | 42 |
| 3.3.3 | Estructura Gestionada | 42 |
| 4 | Conclusiones..... | 43 |
| 4.1 | Cobertura de Objetivos | 43 |
| 4.1.1 | Estructura Mínima..... | 43 |
| 4.1.2 | Estructura Gestionada | 43 |
| 4.2 | Arquitectura Óptima..... | 44 |
| 4.3 | Conclusiones | 44 |
| 4.4 | Vías de Investigación Futuras..... | 45 |
| | Bibliografía..... | 47 |

Índice de figuras

| | |
|--|----|
| Ilustración 1 - Arquitectura Nodo Computación | 10 |
| Ilustración 2 - Arquitectura Nodo Computación OpenSource | 11 |
| Ilustración 3 - Apache Spark: Componentes | 15 |
| Ilustración 4 - Apache Hadoop: Componentes | 17 |
| Ilustración 5 - Apache Hadoop: Arquitectura | 19 |
| Ilustración 6 - Apache Hadoop: YARN mapReduce..... | 20 |
| Ilustración 7 - Sahara: Diagrama de Componentes | 23 |
| Ilustración 8 - Sahara: Escalar Clúster | 24 |
| Ilustración 9 - Sahara: Creación de Trabajos | 25 |
| Ilustración 10 - Sahara: Creación de Plantilla de Grupos de Nodos..... | 26 |
| Ilustración 11 - Sahara: Creación de Plantilla de Clúster | 27 |
| Ilustración 12 - Sahara: Detalles de Clúster | 28 |
| Ilustración 13 - OpenStack: Arquitectura Neutron..... | 35 |
| Ilustración 14 - OpenStack: Arquitectura | 36 |
| Ilustración 15 - Estructura Mínima..... | 42 |
| Ilustración 16 - Estructura Gestionada | 42 |

Siglas

API: Interfaz de Programación de Aplicaciones (Application Programming Interface)

DSR: Enrutamiento de Recursos Dinámico (Dynamic Resource Routing)

EMR: MapReduce Elástico (Elastic MapReduce)

FIFO: Primero en Entrar, Primero en Salir (First In First Out)

GUI: Interfaz Gráfica de Usuario (Graphical User Interface)

HA: Alta Disponibilidad (High Availability)

HDFS: Sistema de Archivos Distribuido Hadoop (Hadoop Distributed FileSystem)

HPC: Computación de Alto Rendimiento (High Performance Computing)

HSM: Administración de Almacenamiento Jerárquico (Hierarchical Storage Management)

IaaS: Infraestructura como Servicio (Infrastructure As A Service)

MPLS: Conmutación Multiprotocolo mediante Etiquetas (Multiprotocol Label Switching)

NIC: Tarjeta de Red (Network Interface Card)

QA: Garantía de Calidad (Quality Assurance)

RAID: Conjunto Redundante de Discos Independientes (Redundant Array of Independent Disks)

REST: Transferencia de Estado Representacional (Representational State Transfer)

RPC: Llamada a procedimiento remoto (Remote Procedure Call)

TCG: Generador de Código Diminuto (Tiny Code Generator)

UI: Interfaz de Usuario (User Interface)

VPN: Red Privada Virtual (Virtual Private Network)

1 Introducción

Justificación de los motivos de desarrollo del presente trabajo, estudio de los sistemas preexistentes, definición de los objetivos del mismo y finalmente, un breve resumen de la estructura del documento.

1.1 Motivación

Uno de los retos a los que actualmente se enfrenta el procesamiento distribuido es la optimización de la asignación de recursos para máquinas individuales. Atendiendo a una pila de funcionalidades básica para una máquina parte de un clúster genérico dedicado por entero a computación en nube (a partir de ahora llamada nodo), obtendríamos la siguiente estructura:



Ilustración 1 - Arquitectura Nodo Computación

Como observamos a simple vista, la pila de tecnologías es densa y numerosa, y probablemente innecesaria o inadecuada para ciertas cargas de trabajo.

1.2 Sistemas Libres

Analizando los sistemas actualmente funcionales a los que el alumno ha tenido acceso, podemos identificar constantes de funcionamiento y despliegue comunes a todos ellos.

Basándonos en la gráfica del punto anterior y reduciendo nuestras elecciones al uso de soluciones libres en un entorno OpenSource, la estructura clásica de despliegue sería la siguiente:

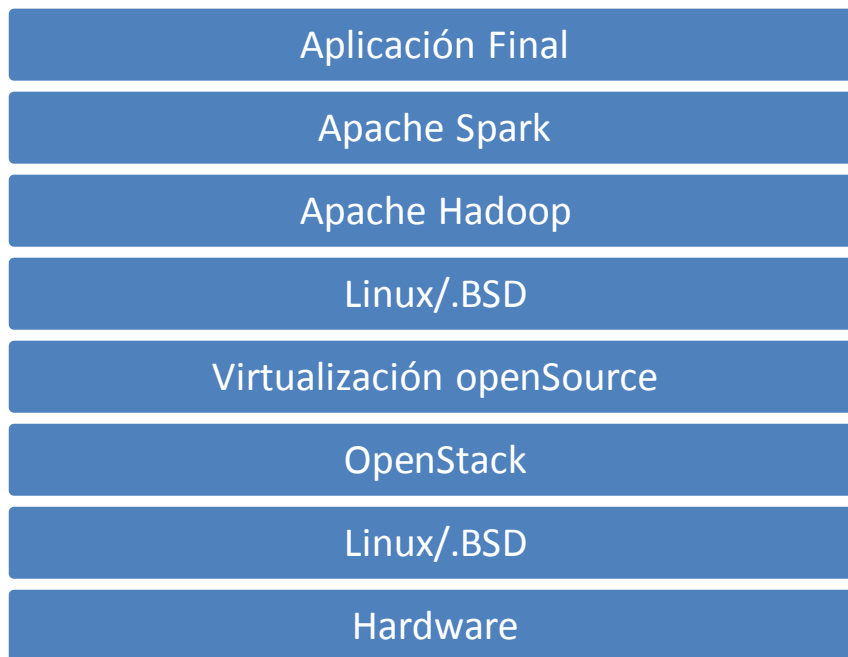


Ilustración 2 - Arquitectura Nodo Computación OpenSource

1.3 Objetivos

Como se observa en las gráficas arriba mostradas, si bien se admite una cierta variación en la selección de componentes, esta pila de procesamiento puede resultar ineficiente para la consecución de la tarea final, la ejecución de aplicaciones cliente.

Se propondrá por tanto en el siguiente trabajo el estudio de las funcionalidades y características de los distintos componentes arriba enumerados, con objeto de buscar redundancias o funciones susceptibles de ser simplificadas, aumentando de esta forma el rendimiento nativo de los nodos individuales, así como del conjunto del sistema.

1.4 Estructura del Documento

En el apartado *Estado del Arte* comenzaremos realizando un estudio pormenorizado de los distintos componentes habituales de una pila de aplicaciones para un nodo típico basado en sistemas OpenSource.

Posteriormente, en el apartado *Análisis de Funcionalidades* identificaremos las funciones propias de cada componente del sistema, en busca de redundancias o funcionalidades superfluas susceptibles de ser eliminadas sin afectar al comportamiento del sistema.

Finalmente, en el apartado Conclusiones, haremos un breve resumen de las mejoras detectadas, susceptibles de ser implementadas, así como de la previsión de mejora del rendimiento del sistema en caso de adopción de las mismas.

2 Estado del Arte

Se intentará aportar en el presente apartado una imagen lo más clara y concisa de los entornos objeto de estudio del trabajo en el momento de realización del mismo.

2.1 Apache Spark[1]

El entorno Apache Spark es un framework de computación clusterizada desarrollado en el AMPLab de la universidad de Berkely. Amplía el paradigma MapReduce basado en disco de dos fases de Hadoop introduciendo primitivas en memoria y permitiendo un grado de aceleración de hasta 100 para ciertas aplicaciones. Dicha estructura lo hace adecuado para operaciones de machine learning en entornos de Big Data.

2.1.1 Características

- Spark provee APIs nativas para los lenguajes de programación Java, Scala y Python.
- Escalabilidad de más de 8000 nodos en producción
- Habilidad de hacer caché de datasets en memoria para análisis de datos interactivo, permitiendo consultas iterativas sobre un mismo working set
- Interfaz de línea de comandos interactiva en Scala o Python para exploración de datos de escala de baja latencia
- Procesado de Streaming mediante librerías de alto nivel.
- Soporte de consultas estructuradas y relacionales SQL
- Librerías de alto nivel para machine learning y procesado visual.

2.1.2 Arquitectura[2]

Spark requiere tanto un administrador de clúster como un sistema de almacenamiento distribuido. Actualmente Spark soporta clústeres nativos, Hadoop YARN o Apache Mesos. Para almacenamiento distribuido las opciones son más amplias, incluyendo HDFS, Cassandra, OpenStack Swift y Amazon S3

Spark Core y Resilient Distributed Datasets (RDDs)

Spark Core es el fundamento del proyecto completo. Programado en java, proporciona las funcionalidades de distribución de tareas, planificación y entrada y salida básicas.

Las RDDs son la abstracción programática fundamental del proyecto Spark, y consisten en una colección lógica de datos particionada en varias máquinas. Pueden generarse referenciando datasets en sistemas de almacenamiento externo o aplicando transformaciones de granulado grueso en RDDs preexistentes.

Se exponen a través de una API integrada en lenguaje nativo Java, Python o Scala, empleándose de forma similar a las colecciones de proceso locales. La complejidad de programación se ve por tanto reducida, dado que la manipulación de RDDs es similar a la de datos locales.

Spark SQL[3]

Reemplazo de las antiguas librerías Shark/Blink. Dedicado al procesamiento de datos estructurados. Provee una capa de abstracción llamada DataFrames, y también puede actuar como un motor de consulta SQL distribuido.

Spark Streaming[4]

Permite el procesamiento de flujos de datos en tiempo real escalable, con alto rendimiento y tolerante a fallos. Los datos se pueden ingerir desde diversas fuentes, procesar mediante algoritmos complejos de alto nivel y finalmente extraer a sistemas de ficheros, bases de datos o paneles en vivo.

MLlib Machine Learning Library[5]

Es la librería escalable de aprendizaje máquina de Spark, y consiste de algoritmos de aprendizaje comunes junto con utilidades, incluyendo clasificación, regresión, clústering, filtrado colaborativo, reducción dimensional y primitivas de optimización subyacentes.

GraphX[6]

Reemplazo de la antigua librería Bagel. Se encarga de la computación de grafos y grafos paralelos. Extiende a alto nivel el RDD introduciendo una nueva abstracción Graph: un multigrafo dirigido con propiedades acopladas a cada vértice y borde. Para permitir la computación de grafos se expone un conjunto de operaciones fundamentales, así como una variante optimizada de la API Pregel. Para simplificar las tareas analíticas, GraphX incluye una colección de algoritmos y constructores de grafos (subgraph, joinVertices, aggregateMessages, ...)

2.1.3 Componentes[7]

Las aplicaciones Spark se ejecutan como grupos de procesos independientes en un clúster, coordinadas por el objeto SparkContext de nuestro programa principal (llamado programa driver). Dicho objeto conecta a varios tipos de administradores de clúster, que distribuyen los recursos entre las aplicaciones. Una vez conectado, Spark adquiere en cada nodo del clúster *ejecutores*, procesos que ejecutan computaciones y

almacenan datos. Después, envía el código de la aplicación a los ejecutores. Finalmente, envía *tareas* para correr en los ejecutores.

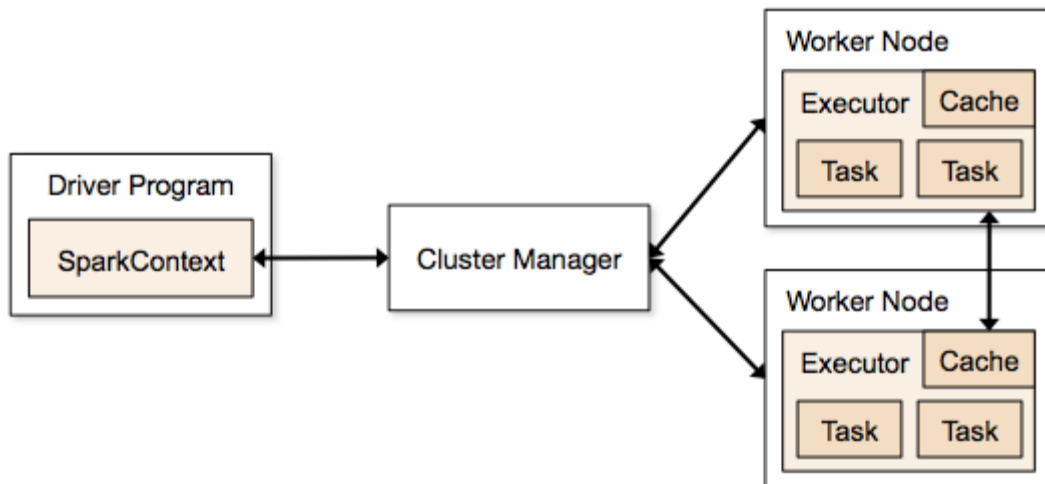


Ilustración 3 - Apache Spark: Componentes

Como notas de interés podemos destacar:

- Cada aplicación obtiene su proceso ejecutor propio, que corre las tareas en múltiples hilos y permanece activo durante todo el tiempo de vida de la aplicación. Esto aísla tanto el control como la ejecución de las aplicaciones, haciendo que la compartición de datos entre distintas aplicaciones deba hacerse a través de almacenamiento externo.
- Spark solo emplea el Administrador de Clúster para adquirir procesos ejecutores. Por tanto su funcionamiento es independiente de la tecnología subyacente o la convivencia con otros entornos de aplicación que puedan correr sobre el mismo administrador.
- Dado que el driver planifica la ejecución de tareas en el clúster, debería correr lo más físicamente cerca posible de los nodos trabajadores. Es preferible el envío de llamadas RPC al driver que su ejecución lejana a los nodos para garantizar la contención de envío de mensajes.

2.2 Apache Hadoop

El entorno Hadoop está escrito en java y está orientado al almacenamiento y procesado distribuido de sets de datos grandes en clústeres de ordenadores desplegados con hardware convencional. Todos los módulos están diseñados de forma que garanticen la tolerancia a fallos.

2.2.1 Características

La filosofía principal de Hadoop es la de mover la computación a los datos, para ello el núcleo de Hadoop consiste en una parte de almacenamiento (HDFS) y una parte de procesado (MapReduce). Hadoop divide ficheros en bloques grandes y los distribuye entre los nodos del cluster. Para procesar los datos de dichos ficheros, mapReduce transfiere el código empaquetado a los nodos para su procesado en paralelo, basado en los datos que cada nodo debe procesar. Esta aproximación permite aprovechar la localidad de los datos (que los nodos manipulen datos de los que disponen en su entorno) para permitir el procesado de forma más rápida y eficiente que en una estructura de supercomputación convencional, que depende de un sistema de ficheros paralelizado donde la computación y los datos están aislados y conectados por redes de alta velocidad.

Detallaremos a continuación los módulos principales del entorno:

- **Hadoop Common:** contiene las librerías y utilidades empleadas por otros módulos Hadoop
- **Hadoop Distributed File System:** un sistema de ficheros distribuido que guarda datos en máquinas de uso convencional, permitiendo anchos de banda agregados muy altos en el clúster
- **Hadoop YARN:** una plataforma de administración de recursos responsable de administrar los recursos de computación en los clústeres y de usarlos para planificar las aplicaciones de usuario.
- **Hadoop mapReduce:** un modelo de programación para procesado de datos de gran escala. Actualmente reemplazado por YARN.

Cabe señalar que actualmente se emplea el termino Hadoop para referirse al ecosistema Hadoop entero, que incluye paquetes adicionales como Apache Pig, Apache Hive, Apache Hbase, Apache Spark y otros.

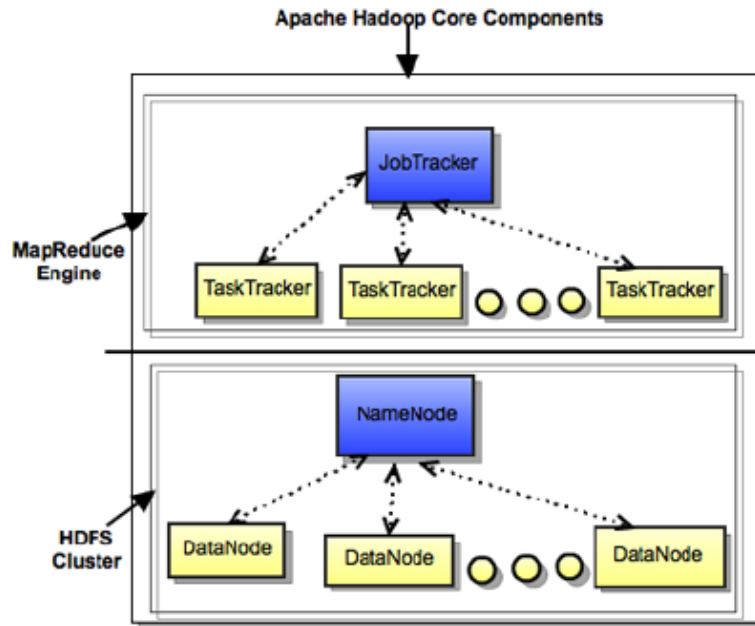


Ilustración 4 - Apache Hadoop: Componentes

2.2.2 Arquitectura

Sistema de ficheros

El sistema HDFS es un sistema de ficheros escalable, distribuido y portable escrito en java para el entorno Hadoop, un clúster Hadoop tiene nominalmente un solo namenode junto con un clúster de datanodes, aunque hay disponibles opciones de redundancia para el namenode dado su rol crítico. Cada datanode sirve bloques de datos por la red usando un protocolo de bloques específico de HDFS, el sistema de ficheros usa sockets TCP/IP para comunicarse, los clientes usan llamadas a procedimiento remoto para comunicarse entre ellos.

HDFS almacena ficheros grandes sobre varias máquinas, esto permite fiabilidad replicando los datos en múltiples servidores, por lo que no se requiere de RAID en los hosts. El valor de replicado por defecto es 3, con lo que los datos se almacenan en tres nodos, dos en el rack local y uno en un rack diferente. Los nodos de datos pueden comunicarse entre ellos para redistribuir los datos, mover copias y mantener la replicación alta.

El namenode secundario no es una réplica del primario, sino un conector que hace copias de seguridad de la información de directorio del primario, que el sistema guarda a directorios externos. Estas imágenes permiten relanzar un namenode primario fallido sin tener que reejecutar la secuencia de acciones del sistema de ficheros y editar el registro de acciones.

Por ser el namenode el único punto de almacenamiento y administración de metadatos, puede ser un cuello de botella cuando se trata un número elevado de ficheros, especialmente muchos pequeños ficheros.

Es de señalar también la compartición de conocimiento entre el jobTracker y el taskTracker, lo que permite reducir el tráfico que viaja por la red y prevenir transferencias innecesarias de datos. Con otros sistemas de ficheros, no “rack-aware” esta característica puede no estar disponible.

HDFS está diseñado principalmente para ficheros inmutables, por lo que no se adecúa a sistemas que requieran operaciones de escritura concurrentes.

Motor MapReduce (YARN)

Consiste en un jobTracker, al que las aplicaciones cliente envían trabajos mapReduce. El jobTracker envía los trabajos a los nodos taskTrackers disponibles en el clúster, intentando mantener el trabajo tan cerca de los datos como sea posible. Con un sistema de ficheros “rack-aware”, el jobTracker sabe que nodo contiene los datos y qué otras máquinas están cerca. Si el trabajo no se puede alojar en el nodo donde residen los datos, se le da prioridad a nodos del mismo rack. Esto reduce el tráfico de red de la red principal. Si un taskTracker falla y supera el tiempo de espera, esa parte del trabajo se replanifica. El taskTracker de cada nodo lanza un proceso de máquina virtual java separada para prevenir que él mismo falle si el trabajo en ejecución cuelga su máquina. Se envía una señal de heartbeat del taskTracker al jobTracker cada varios minutos para comprobar su estado.

Sobre los mecanismos de planificación del motor, por defecto se incluye una cola FIFO de procesado y opcionalmente 5 prioridades. No hay desalojado una vez que un trabajo está corriendo. Sin embargo ha habido intentos adaptados a usos específicos que se consideran interesantes y se detallan a continuación:

- Fair Scheduler: Desarrollado por Facebook. En este planificador los trabajos se agrupan en pools, tras lo cual a cada pool se le asigna una partición mínima garantizada. Finalmente la capacidad en exceso se divide entre trabajos.
- Capacity Scheduler: Desarrollado por Yahoo. En este planificador los trabajos se organizan también en colas, repartiéndose éstas una fracción de la capacidad de recursos total. Los recursos libres se distribuyen en colas más allá de su capacidad total. Es de señalar que en este sistema, en una cola un trabajo con un alto nivel de prioridad tiene acceso a los recursos de la cola.

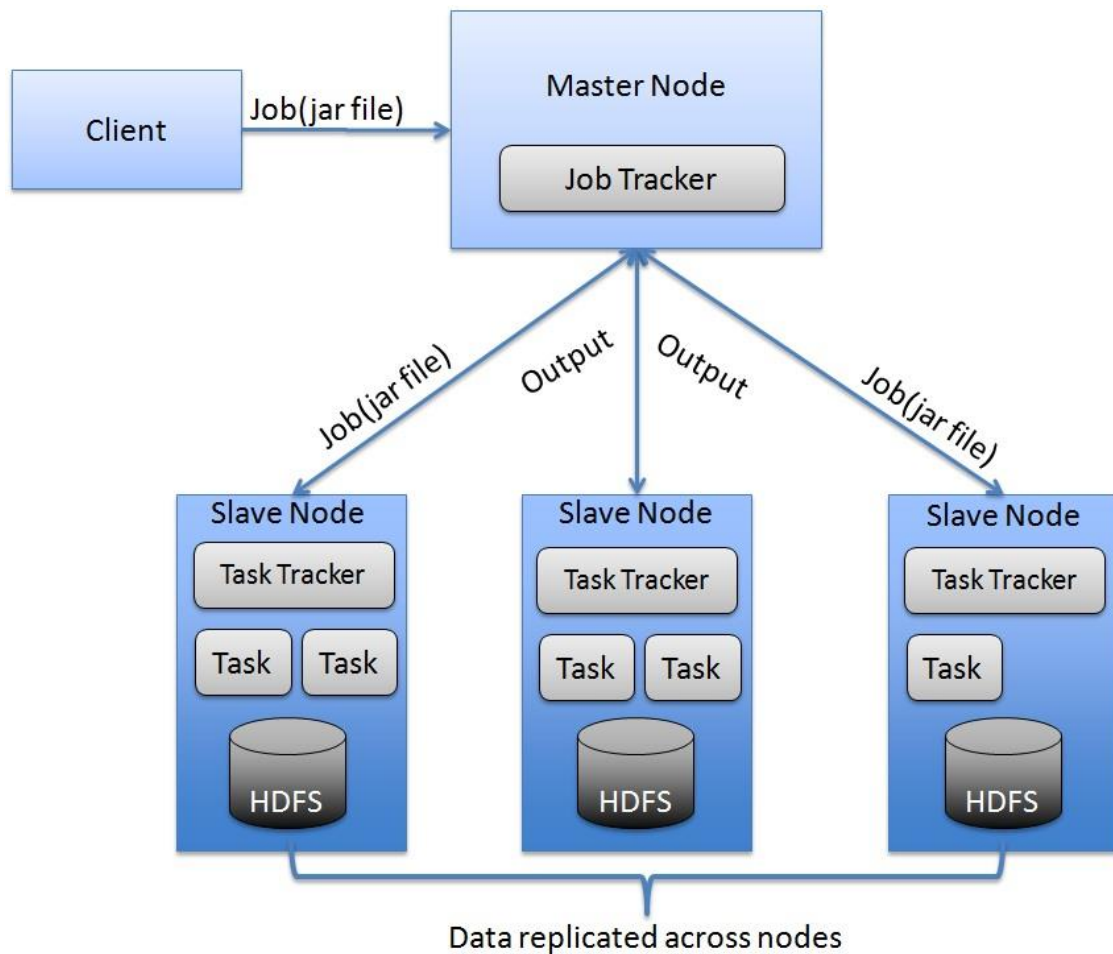


Ilustración 5 - Apache Hadoop: Arquitectura

2.2.3 Componentes[8]

Sistema de Ficheros

Como ya hemos visto, HDFS crea múltiples réplicas de cada bloque de datos y las distribuye en clústeres para permitir acceso rápido y fiable. Para ello implementa los siguientes componentes:

- **Namenode**: Es el componente principal del sistema. Se encarga de mantener el sistema de nombres y administrar los bloques que estarán presentes en los Datanodes.
- **Datanodes**: son los sistemas “esclavos” que se despliegan en cada máquina y que proveen del almacenamiento efectivo. Son responsables de atender peticiones de lectura y escritura de los clientes.
- **Namenode Secundario**: Es el responsable de realizar comprobaciones periódicas. En caso de fallo de un Namenode, se puede reiniciar el mismo a partir de su punto de comprobación.

MapReduce

El entorno mapReduce usa el paradigma computacional del mismo nombre para el procesado distribuido de datos. En dicho entorno, cada trabajo tiene una fase de mapeado asignada por el usuario (que es un procesado de la entrada paralelo y sin compartición), seguida por una fase de reducción definida por el usuario en la que la salida del mapa se agrega. De forma típica, HDFS será el sistema predeterminado de entrada y salida para operaciones de tipo map-reduce.

Los componentes principales de esta estructura son los siguientes:

- **JobTracker:** Es el elemento principal del sistema que administra los trabajos y recursos en el clúster. El JobTracker intenta planificar cada mapa tan cerca de los datos reales que se están procesando como sea, posible, es decir, en el TaskTracker que corre en el mismo datanode que el bloque subyacente.
- **TaskTrackers:** son los “esclavos que se despliegan en cada máquina. Son responsables de ejecutar las tareas de mapeo y reducción bajo las instrucciones del jobTracker.
- **JobHistoryServer:** es un daemon que sirve información histórica sobre las aplicaciones completadas. Típicamente puede desplegarse de forma concurrente con el jobTracker, pero es recomendable ejecutarlo por separado.

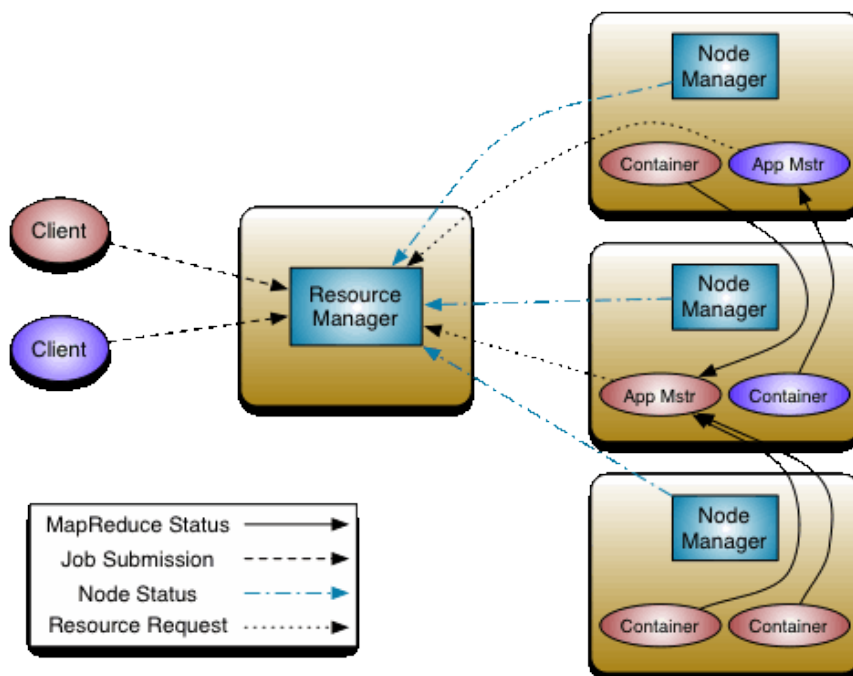


Ilustración 6 - Apache Hadoop: YARN mapReduce

2.3 OpenStack Sahara[9]

El proyecto OpenStack Sahara (antes Savanna) comenzó su vida bajo el entorno Apache 2.0, para después integrarse en OpenStack. Se desarrolla como opensource con intervención de las fundaciones Apache, OpenStack, Mirantis, Hortonworks y RedHat.

2.3.1 Características[10]

El entorno Apache Hadoop arriba visto es una implementación de MapReduce estándar de la industria y ampliamente adoptado, por ejemplo en el servicio de Amazon Elastic MapReduce (EMR). Como solución de software libre, es un producto apropiado para la solución de problemas de analítica de Big Data, pero su intensidad de uso de recursos, demanda agilidad de escalado creciente/decreciente, lo cual es difícil dado la complejidad de despliegue, pruebas, optimización y mantenimiento. Es por tanto razonable el ofrecer aprovisionamiento y administración simples de clústeres de máquinas Hadoop en OpenStack, con el objetivo de ofrecer Hadoop Elástico Bajo Demanda.[11]

La finalidad del proyecto OpenStack Sahara es proveer a los usuarios con medios simples para provisionar un clúster Hadoop en OpenStack especificando varios parámetros tales como:

- Versión de Hadoop
- Topología del clúster
- Detalles de hardware de los nodos

Tras la introducción de dichos parámetros por parte del usuario, es posible para Sahara desplegar el clúster en cuestión de minutos. Sahara también permite escalar clústeres previamente provisionados permitiendo añadir/eliminar nodos de trabajo bajo demanda.

Casos de uso

La solución Sahara se dedicará a la resolución de los siguientes casos de uso:

- Aprovisionamiento rápido de clústeres Hadoop en OpenStack para entornos de desarrollo y QA
- Utilización de la potencia de computación no empleada de la nube de propósito general OpenStack IaaS
- “Analítica como Servicio” para cargas de trabajo de analítica ad-hoc o en ráfagas (como el EMR previamente citado)

Características clave

Entre las características principales podemos señalar:

- Diseño como componente de OpenStack
- Administrado a través de una API REST con UI disponible como parte del OpenStack Dashboard
- Soporte para diferentes distribuciones de Hadoop
- Sistema de instalación de motores de despliegue de Hadoop mediante plugins
- Integración con herramientas de administración propietarias como Apache Ambari o Consola de Administración Cloudera
- Plantillas predefinidas de configuración de Hadoop con la habilidad de modificar parámetros.

Relación con otros componentes

El producto Sahara comunica con los siguientes componentes OpenStack:

- **Horizon:** provee una GUI con la capacidad de emplear todas las características de Sahara
- **Keystone:** autentifica usuarios y provee tokens de seguridad que se emplean para acceso a recursos OpenStack, permitiendo el limitar el acceso de los usuarios de Sahara a recursos del sistema basado en sus privilegios
- **Nova:** Empleado para provisionar máquinas virtuales para un clúster Hadoop
- **Heat:** Orquestador para los servicios requeridos por el clúster Hadoop. De uso opcional.
- **Glance:** Almacén de las imágenes de VMs Hadoop, cada una de ellas conteniendo un sistema operativo instalado y Hadoop
- **Swift:** Opcionalmente puede emplearse como almacén para los datos procesados por los trabajos Hadoop
- **Cinder:** opcionalmente puede emplearse para el almacenamiento de bloques
- **Neutron:** provee el servicio de red
- **Ceilometer:** empleado para recopilar estadísticas de uso del clúster con motivo de monitorización y medición del rendimiento del mismo

Scale Cluster ×

+

| Group Name | Template | Count |
|--|--------------------------------------|--|
| <input type="text" value="master"/> | <input type="text" value="master"/> | 1 - + |
| <input type="text" value="storage"/> | <input type="text" value="storage"/> | 5 - + |
| <input type="text" value="worker"/> | <input type="text" value="worker"/> | 0 - + |
| <input type="text" value="new-compute"/> | <input type="text" value="compute"/> | 7 - + Remove |

Ilustración 8 - Sahara: Escalar Clúster

Analítica bajo demanda

Para procesos de analítica como servicio, el flujo de trabajo será el siguiente:

- Seleccionar una de las versiones predefinidas de Hadoop
- Configurar el trabajo:
- Seleccionar el tipo de trabajo: pig, hive, fichero-jar, etc
- Provisionar la fuente del script de trabajo o localización del jar
- Seleccionar localización de datos de entrada y salida
- Seleccionar la localización de los registros
- Seleccionar límites para el tamaño de clúster
- Ejecutar el trabajo
- Todos los procesos de provisionado y ejecución del trabajo serán transparentes para el usuario
- El clúster se eliminará automáticamente cuando se complete el trabajo
- Obtener los resultados de la computación

Create Job Binary ×

Name *

Storage type *

Internal database

Internal binary

*Upload a new file

Upload File

fun.pig

Description

Important: The name that you give your job binary will be the name used in your job execution. If your binary requires a particular name or extension (ie: ".jar"), be sure to include it here.

Select the storage type for your job binary.

- Data Processing internal database
- Swift

For Data Processing internal job binaries, you may choose from the following:

- Choose an existing file
- Upload a new file
- Create a script to be uploaded dynamically

For Object Store job binaries, you must:

- Enter the URL for the file
- Enter the username and password required to access that file

You may also enter an optional description for your job binary.

Ilustración 9 - Sahara: Creación de Trabajos

2.3.3 Componentes[12]

Autenticación y Autorización

La API de Sahara emplea el servicio de identidad Keystone como su servicio de autenticación por defecto. Cuando Keystone está habilitado, los usuarios que hacen peticiones al servicio Sahara deben proveer un token de autenticación, que pueden obtener autenticándose con el endpoint Keystone.

La autorización se realiza a nivel de tenant, por lo que Sahara realizará las operaciones solicitadas sobre el tenant especificado usando las credenciales provistas. Por tanto los clústeres solo pueden ser creados y administrados sobre tenants a los que el usuario tenga acceso.

Plugins

Los objetos de tipo plugin informan sobre qué versión/distribución de Hadoop pueden instalar, así de qué configuraciones se pueden emplear en el clúster.

Registro de Imágenes

El registro de imágenes es una herramienta para la administración de imágenes de máquinas virtuales. Cada plugin provee de una lista de marcadores de los que la imagen debería disponer. Sahara también requiere un nombre de usuario para permitir el acceso a una instancia de sistema operativo para la ejecución de operaciones remotas.

El registro de imágenes también provee de la habilidad de añadir etiquetas a las imágenes, así como definir el nombre de usuario de sistema operativo.

Plantillas de Grupos de Nodos

Una plantilla de grupos de nodos describe un grupo de nodos en un clúster. Contiene una lista de procesos de Hadoop que serán lanzados en cada instancia en un grupo. También puede proveer de una configuración limitada al nodo para dichos procesos. Este tipo de plantillas encapsulan los parámetros hardware (sabor) para la vm del nodo y la configuración para procesos Hadoop que corren en el nodo.

Create Node Group Template

Configure Node Group Template *

Template Name *

Description

OpenStack Flavor *

m1.micro

Storage location * ?

Ephemeral Drive

Floating IP pool

Do not assign floating IPs

Auto Security Group ?

This Node Group Template will be created for:
Plugin: cdh
Hadoop version: 5

The Node Group Template object should specify processes that will be launched on each instance. Also an OpenStack flavor is required to boot VMs.

Data Processing provides different storage location options. You may choose Ephemeral Drive or a Cinder Volume to be attached to instances.

When processes are selected, you may set **node** scoped Hadoop configurations on corresponding tabs.

Ilustración 10 – Sahara: Creación de Plantilla de Grupos de Nodos

Plantillas de Clústeres

Una plantilla de clúster se diseña para unir plantillas de grupos de nodos para formar un clúster. Una plantilla de clúster, define por tanto qué grupos de nodos se incluirán y cuántas instancias de los mismos se crearán en cada uno. Algunas de las configuraciones de Hadoop no se pueden aplicar a un solo nodo, sino a un clúster completo, de forma que los usuarios pueden implementar las mismas en una plantilla de clúster. Sahara permite a los usuarios especificar qué procesos deberían añadirse a un grupo anti-afinidad dentro de una plantilla de clúster. Si se incluye un proceso en un grupo anti-afinidad, quiere decir que las vms donde este proceso se lance deberán planificarse para su despliegue en distintos anfitriones hardware

| Group Name | Template | Count | |
|-------------|-------------|-------|------------|
| cdh-master | cdh-master | 1 | - + Remove |
| cdh-worker | cdh-worker | 3 | - + Remove |
| cdh-manager | cdh-manager | 1 | - + Remove |

Ilustración 11 - Sahara: Creación de Plantilla de Clúster

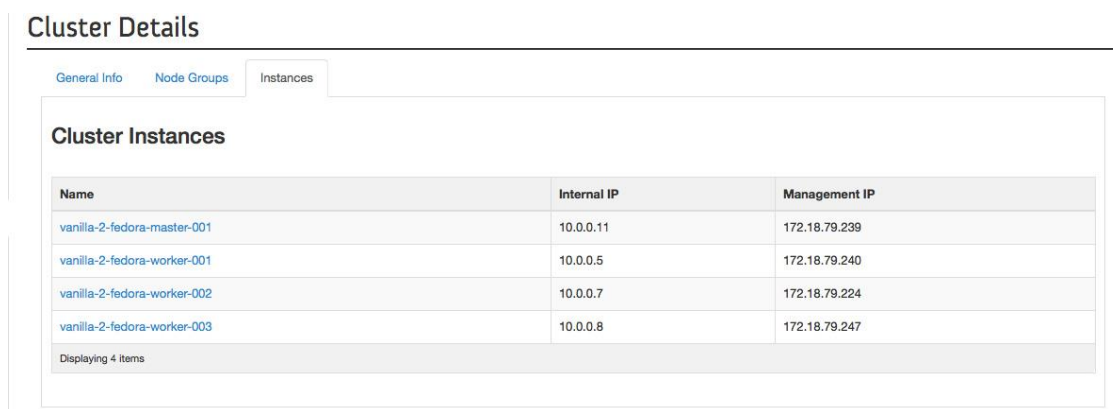
Clústeres

La entidad clúster representa un clúster Hadoop. Se caracteriza principalmente por una imagen de vm con un Hadoop preinstalado que se usara para el despliegue del clúster. El usuario podrá elegir una plantilla predeterminada del clúster para arrancar un clúster. Para acceder a las vms tras el comienzo del clúster, el usuario deberá especificar un par-clave.

Sahara provee de varias restricciones en la topología de clúster Hadoop. Los procesos JobTracker y NameNode pueden ejecutarse en una sola vm o en dos separadas. El clúster también puede contener nodos de trabajo de diferentes tipos. Los nodos de trabajo pueden ejecutar tanto TaskTracker y Datanode de forma simultanea

como cualquiera de ambos por separado. Sahara permite a los usuarios crear un clúster con cualquier combinación de estas opciones, pero no permitirá la creación de una topología no funcional, por ejemplo un conjunto de workers con DataNodes pero sin un NameNode.

Cada clúster pertenece a un inquilino (tenant) determinado por el usuario. Los usuarios solo tienen acceso a objetos localizados en los inquilinos a los que tienen acceso. Los usuarios solo pueden editar/eliminar los objetos que han creado. Los usuarios administradores, naturalmente tienen acceso a cualquier objeto. Esta es la forma en la que Sahara cumple con la política de acceso general de OpenStack



The screenshot shows the 'Cluster Details' page in Sahara. It has three tabs: 'General Info', 'Node Groups', and 'Instances'. The 'Instances' tab is active, displaying a table titled 'Cluster Instances'. The table has three columns: 'Name', 'Internal IP', and 'Management IP'. It lists four instances: a master node and three worker nodes. At the bottom of the table, it says 'Displaying 4 items'.

| Name | Internal IP | Management IP |
|-----------------------------|-------------|---------------|
| vanilla-2-fedora-master-001 | 10.0.0.11 | 172.18.79.239 |
| vanilla-2-fedora-worker-001 | 10.0.0.5 | 172.18.79.240 |
| vanilla-2-fedora-worker-002 | 10.0.0.7 | 172.18.79.224 |
| vanilla-2-fedora-worker-003 | 10.0.0.8 | 172.18.79.247 |

Ilustración 12 - Sahara: Detalles de Clúster

2.3.4 Herramientas Opcionales

Integración con Swift

Como ya se ha visto, el servicio Swift proporciona el almacenamiento de objetos de forma estándar en un entorno OpenStack, de forma análoga a Amazon S3. Por lo general se despliega en máquinas bare-metal. Se espera por tanto que Hadoop sobre OpenStack procese sus datos almacenados en Swift, por lo que se proponen varias mejoras al sistema:

- Primero, la implementación de un Sistema de Ficheros para Swift (HADOOP-8545), que haría compatible los entornos Swift con HDFS.
- En segundo lugar, la habilidad de Swift de listar endpoints para un objeto, cuenta o contenedor, haciendo posible la integración del mismo con software que depende de la información de localización de datos para evitar la sobrecarga de red.

Monitorización y Despliegue bajo demanda

Además de las capacidades provistas por herramientas de administración específicas del vendedor Hadoop, Sahara permitirá la integración mediante plugins bajo demanda de sistemas externos de monitorización, como Nagios o Zabbix.

Tanto las herramientas de despliegue como de monitorización se instalarán en máquinas virtuales aisladas, permitiendo que una sola instancia controle varios clústeres de forma simultánea.

2.4 Virtualización OpenSource

2.4.1 Virtualización Hardware

Incluiremos en esta categoría a los sistemas que emulan completamente el hardware del sistema a emular; por lo tanto permiten la ejecución de sistemas operativos invitados distintos del del anfitrión sobre el mismo hardware. Estos sistemas también reciben el nombre de hipervisores.

Linux Xen[13]

Desarrollado por el laboratorio de computación de la universidad de Cambridge, el entorno xen está basado en un diseño de microkernel. Denomina a sus máquinas virtuales dominio, y basa la administración de memoria y cpu de las mismas en el uso de un dominio de máximo privilegio (dom0) que es el único con acceso a todos los recursos hardware de la máquina anfitrión. Dicho dominio es gestionado por el hipervisor, y permite el despliegue de máquinas virtuales invitadas (en la nomenclatura xen, dominios de usuario sin privilegios o domU).

Los dominios de usuario pueden tener acceso a instrucciones privilegiadas mediante tecnologías de virtualización hardware propias del procesador host, como Intel VT-x o AMD-V o mediante técnicas de paravirtualización que permiten el reemplazo de dichas instrucciones con llamadas directas al hipervisor. Estas técnicas requieren la modificación del sistema invitado.

Actualmente se dispone de varias consolas de administración dedicadas a la configuración, arranque, modificación y parada de invitados xen, si bien todas ellas son desarrollos de terceras partes, no dependientes del proyecto original.

VirtualBox[14]

Desarrollado originalmente por Innotek, y pasando después por Sun y Oracle, en 2010 se liberó sobre licencia GPLv2. Emplea las tecnologías de virtualización de Intel y AMD arriba vistas, si bien también permite la virtualización basada en software, en la que modifica al vuelo el código del sistema operativo invitado para ejecutarse en el ring1 de la arquitectura ring de Intel. El código de invitado de modo usuario corre en ring 3, y generalmente accede directamente al hardware del host.

VirtualBox también contiene un recompilador dinámico basado en Qemu para modificar al vuelo cualquier código que intente ejecutarse en modo real o modo protegido, como es el caso de los sistemas BIOS, DOS o de arranque de sistemas operativos invitados.

Con respecto a los dispositivos disponibles para el invitado, VirtualBox ofrece drivers paravirtualizados para discos duros, tarjetas de red, gráficas y de audio y otros

componentes básicos del sistema, instalados como extensiones del invitado y que permiten mejorar el rendimiento de ejecución del mismo.

KVM[15]

Kernel-based Virtual Machine, es una infraestructura para el kernel linux integrada en el mismo desde 2007 y que permite emplearlo como hipervisor. Requiere el soporte de extensiones de virtualización en el procesador host y se integra como un módulo independiente de dicho kernel.

KVM no realiza emulación de hardware, simplemente expone una interfaz de dispositivo desde la que un host de espacio de usuario puede configurar el espacio de direcciones del invitado, derivar la entrada y salida del mismo y mapear los displays de vuelta al anfitrión. El anfitrión de espacio de usuario más usado es QEMU, que veremos a continuación

KVM incluye soporte para paravirtualización de dispositivos en los sistemas invitados, incluyendo tarjeta de red, disco, gráfica y un dispositivo de tipo globo para ajustar el uso de memoria.

QEMU[16]

QuickEmulator, es un hipervisor que implemente virtualización de hardware, diseñado como monitor de máquina virtual alojada. Para ello realiza traducción binaria dinámica, pudiendo ejecutar los sistemas invitados sin modificar.

QEMU admite distintos modos de operación, que enumeraremos a continuación:

- Emulación de modo usuario: emulador de CPU para procesos de usuario, permitiendo la ejecución de aplicaciones de arquitecturas distintas de la nativa.
- Emulación de sistema: Virtualización completa del sistema invitado junto con su hardware correspondiente. Permite la emulación de sets de instrucciones de arquitecturas diversas.
- Hospedado KVM: Junto con KVM y disponiendo de extensiones de virtualización hardware, puede correr máquinas virtuales a velocidades cercanas a la nativa. Aquí QEMU gestiona la migración y configuración de imágenes así como la emulación de hardware, pero la ejecución del invitado la realiza KVM.
- Hospedado Xen: Aquí QEMU se encarga de proveer la capa de emulación de hardware. La ejecución del invitado la realiza íntegramente Xen y es inaccesible para QEMU.

Entre las características reseñables de QEMU podemos destacar su ejecución íntegra en modo usuario, así como la paravirtualización de hardware o integración de servicios de comunicación directa entre el anfitrión e invitado. De especial interés es el sistema TCG, o Tiny Code Generator, que permite abstraer el código fuente de arquitecturas específicas reescribiéndolo en una notación intermedia independiente de máquina. TGC requiere la escritura de código portable dedicado así como la reescritura de la traducción de instrucciones del sistema destino.

2.4.2 Virtualización de Sistema Operativo

Sistemas que mantienen un kernel común entre máquinas y virtualizan los grupos y espacio de nombres de las distintas máquinas; por lo tanto no permiten la ejecución de sistemas operativos distintos del del anfitrión. También llamada virtualización ligera o virtualización de contenedor.

LXC[17]

Diminutivo de LinuxContainers, aprovecha la funcionalidad de *cgroups*, presente en el kernel Linux desde la versión 2.6.24, que permite la limitación o priorización de recursos como CPU, memoria, I/O o red sin necesidad de virtualización. Esto junto con la funcionalidad de aislamiento de espacio de nombres permite garantizar la seguridad y aislamiento de la aplicación del sistema operativo subyacente. Además, dado que se basa en tecnologías propias del kernel oficial, no requiere de parches o modificaciones al mismo, funcionando en cualquier implementación de forma nativa.

OpenVZ[18]

Open Virtuozzo basado en el producto comercial Virtuozzo, renombra a los contenedores como virtual private servers o virtual environments. Se basa en un kernel opcional personalizado y herramientas de línea de comandos. Como diferencia a LXC, cabe destacar el uso de User Beancounters, contadores, límites y garantías que permiten evitar que un solo contenedor monopolice todos los recursos del sistema, así como la funcionalidad de Checkpoint y live migration, que permite el movimiento en caliente de contenedores de un anfitrión a otro con un mínimo tiempo de baja.

2.5 OpenStack [19]

Se describirá brevemente a continuación el entorno OpenStack, atendiendo a una descripción general de las posibilidades del mismo para después hacer un desglose pormenorizado de sus principales componentes.

2.5.1 Características [20]

El entorno OpenStack se propone como plataforma de software libre y abierto para computación en nube, estructurada como Infraestructura como Servicio (IaaS)

Es un error pensar en OpenStack como un producto. Sería más acertado enfocarlo como una fundación independiente, sin ánimo de lucro, que pretende recoger varios proyectos de código abierto, de forma similar a la fundación Apache. La diferencia con ésta es que los productos OpenStack están orientados exclusivamente a IaaS (Infraestructura como Servicio).

- Así mismo, las funcionalidades de OpenStack no son estáticas como en el caso de VMware. Los proyectos se dividen en *core* (del núcleo principal), *incubated* (aprobados por la comunidad para inclusión en la próxima revisión del núcleo principal) y *community* (ni aprobados ni soportados por la comunidad sino posibles fuentes de nuevos proyectos)
- Está estructurado como un conjunto de sistemas individuales interrelacionados y gestionados como proyectos independientes, cada uno de ellos especializado en una tarea determinada. Las tres tareas principales ofrecidas por el entorno son las siguientes:[2]
- **Computación:** Ofrecida por el nodo OSCompute (Nova), al que da soporte OSImage (Glance).
- **Redes:** Aportado por OSNetworking (Neutron)
- **Almacenamiento:** Subdividido en almacenamiento de bloques convencional con OSBlock Storage (Cinder) o almacenamiento de objetos OS con Swift.

2.5.2 Arquitectura

Detallaremos a continuación los módulos principales que configuran la arquitectura mínima OpenStack.

Servicio de Identidad

Encargado de trazar los usuarios y sus servicios asociados, así como de proveer de un listado de servicios y los endpoints de sus APIs respectivas.

Servicio de Imágenes

Encargado del descubrimiento, registro y obtención de imágenes de máquinas virtuales. Ofrece una API RESTful para la consulta de metadatos y adquisición de la imagen final. Permite el almacenamiento de dichas imágenes tanto en sistema de ficheros local como en OS Object Storage

Servicio de Computación

Encargado de alojar y administrar sistemas de computación en nube. Los módulos principales están implementados en Python. Interactúa con el servicio de imágenes, el de identidad y el Panel de control. Puede escalar horizontalmente en hardware estándar.

Componentes de Red

Se ofrecen aquí dos opciones dependiendo de la complejidad de la red a desplegar.

Por una parte el servicio nova-network ofrece funcionalidades para redes básicas, permitiendo un tipo de red por instancia.

El servicio OS Networking (Neutron), anteriormente conocido como Quantum, por el contrario, ofrece múltiples redes por instancia.

Su objetivo es proveer de una abstracción de red escalable, bajo demanda y agnóstica de tecnología, implementando el concepto de “red como servicio” entre interfaces de dispositivos gestionados por otros servicios de OpenStack.

Para ello, Neutron ofrece las siguientes características:

- Abstracción de la API para redes virtuales: segmentos de red básicos de nivel 2
- Abstracción de la API para puertos virtuales: puntos de interconexión para dispositivos que conectan a redes virtuales. Análogos a los vistos en casos anteriores.
- Interacción con otros servicios OpenStack: mediante la asociación de puertos virtuales a vNICs con “servidores virtuales”
- Soporte de distintas tecnologías de backend: plugins para Open vSwitch, Cisco UCS, Linux Bridge, Nicira NVP, Ryu controller, VMware NSX
- Extensibilidad de la API para características específicas del backend: Calidad de servicio, estadísticas por puerto, grupos de seguridad, etc.

La arquitectura de plugins de Neutron le permite la creación y asociación de redes de forma dinámica. Para ello divide sus funcionalidades en dos frentes principales:

- Notificaciones a la API de procesamiento: almacenando los resultados de todas las llamadas a red y puerto además de mapear entidades abstractas a específicas del plugin.
- Administración de vSwitches: mediante mecanismos de autodescubrimiento y autoconfiguración que permiten una interacción mínima con el sistema en caso de cambio-inserción de un nuevo switch.

Quantum Architecture (adv.)

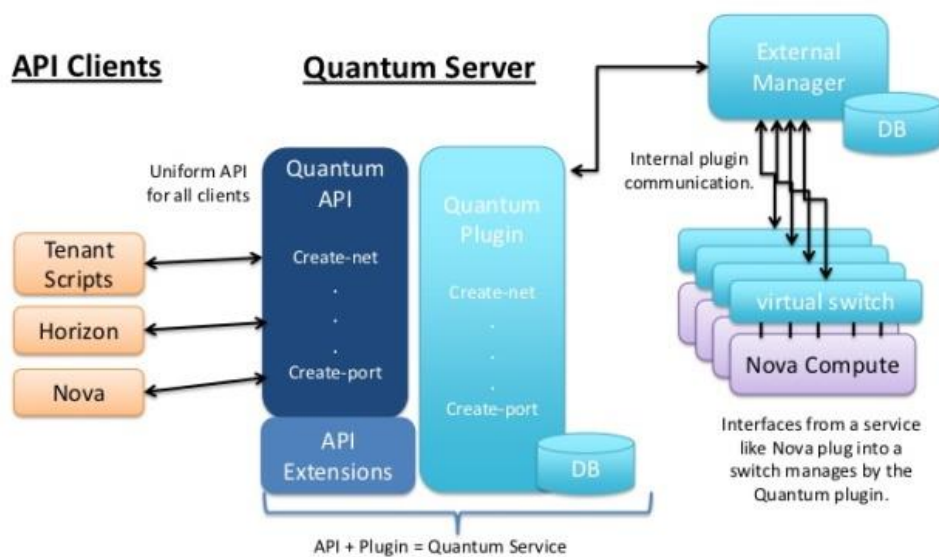


Ilustración 13 - OpenStack: Arquitectura Neutron

Panel de Control (Dashboard)

Proporciona una interfaz gráfica de usuario para acceder, provisionar y automatizar los recursos basados en nube. De diseño extensible y basado en plugins, permite integrar servicios y productos de terceros, tales como herramientas de administración adicionales.

Servicio de Almacenamiento

Se ofrecen las opciones de almacenamiento de bloques, para exponer y conectar a instancias de computación dispositivos de bloques de datos clásicos, junto con el almacenamiento de objetos, que ofrece una plataforma distribuida y accesible por api que se puede integrar directamente en las aplicaciones o usarse para backup, archivado o retención de datos.

Otros módulos

De menor interés para nuestro caso de estudio, pero también dignos de mención son el módulo de orquestación, el de telemetría o el servicio de Bases de Datos

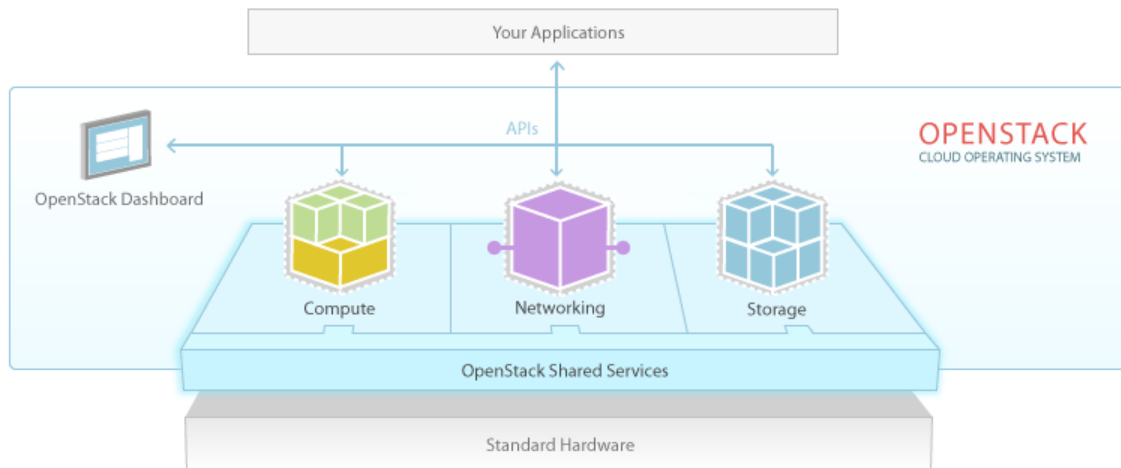


Ilustración 14 - OpenStack: Arquitectura

2.5.3 Componentes [21]

Se detallan a continuación muy brevemente los componentes principales ofrecidos por el sistema OpenStack, así como la conceptualización de uso asociada a cada uno de ellos:

- **Usuario:** representación de cualquier persona, sistema o servicio que usa un servicio de OS.
- **Credencial:** Datos que confirman la identidad del usuario.
- **Autenticación:** Proceso de confirmación de la identidad de un usuario validando sus credenciales.
- **Token:** Cadena de texto alfanumérica que permite el acceso a APIs y servicios de OS.
- **Localizador (Tenant):** Contenedor usado para agrupar o aislar recursos.
- **Servicio:** Cada uno de los servicios de OS.
- **Endpoint:** Dirección accesible por red propia de un servicio, usualmente una URL.
- **Rol:** Una personalidad con un conjunto determinado de permisos y privilegios que permiten el acceso a ciertas operaciones de cada servidor.

Incluido en el token de usuario.

- **Ciente Principal (Keystone):** Interfaz de línea de comando para gestión de la API de identidad.

3 Análisis de Funcionalidades

Se desglosarán a continuación las funcionalidades imprescindibles, necesarias y recomendables que debe ofrecer un sistema de procesamiento distribuido, en base a las ideas expuestas en apartados anteriores.

3.1 Usos Finales

Usaremos como base de nuestra organización la guía de diseño de arquitectura ofrecida en la documentación de OpenStack, en la que se ofrecen varias configuraciones por defecto, atendiendo a los casos de uso y despliegues más comunes.[22] Se detallan a continuación los entornos ofrecidos con la intención de cuadrarlos con los objetivos vistos arriba, para lo cual nos basaremos en las recomendaciones de clasificación aportadas por la arquitectura OpenStack:

3.1.1 Propósito General

Se considera un punto de arranque para construir un despliegue en nube. Estos diseños balancean el uso de componentes sin poner énfasis en ninguna funcionalidad concreta. Esta misma elección hace que no sean válidos para situaciones de uso especializadas o implementaciones en casos extremos.

Se recomienda por tanto su uso para aproximaciones de escalado horizontal, que pueden beneficiarse del uso de un mayor número de sistemas estáticos de características similares.

3.1.2 Orientado a Computación

Diseño pensado para cargas computacionales intensas. Se hace hincapié en la necesidad de recursos de CPU y RAM necesarios para soportar dichas cargas. Se reconocen como casos de uso de este tipo las siguientes situaciones:

- Computación de Alto Rendimiento (HPC)
- Análisis de Big Data con almacenamiento de datos distribuido tipo Hadoop o similar.
- Entornos de Integración/Despliegue Continuos (CI/CD)
- Plataforma como Servicio (Platform-as-a-Service o PaaS)
- Procesado de Señal para virtualización de funciones de red (Signal processing for network function virtualization o NFV)

En dichos entornos serán necesarios servicios adicionales de tipo configuración de red, mientras que por el contrario no será habitualmente necesario almacenamiento de bloques persistente.

3.1.3 Orientado a Almacenamiento

Modelo de almacenamiento de datos en el que los datos digitales se guardan en pools lógicas y almacenamiento físico a través de varios servidores y localizaciones. Dicha definición habitualmente responde al almacenamiento de objetos, pero también se ha extendido al almacenamiento de bloques.

Por encima de cierta escala, es necesario implementar sistemas de HSM y data grids para automatizar las decisiones sobre datos. Mientras que los sistemas HSM permiten la gestión y movimiento automático de los datos, así como la orquestación de operaciones de datos, los data grids agrupan conjuntos de servicios encargados de la gestión de datasets de gran tamaño.

3.1.4 Orientado a Red

Todos los sistemas OS dependen hasta algún punto de las comunicaciones de red para su funcionamiento. Hay situaciones, en todo caso, que por diseño son fuertemente dependientes de la infraestructura de red subyacente y en las que tiene sentido dedicar recursos extra a la misma. Podemos destacar:

- Red de distribución de contenidos
- Funciones de administración de red
- Oferta de servicios de red (VPNs, MPLSs, ...)
- Portales o Servicios Web
- Big Data
- Alta Disponibilidad
- Voz sobre IP
- Videoconferencia

3.1.5 Multi-Sitio

Consiste en varios servidores en localizaciones distintas.

3.1.6 Híbrido

Tanto en el sentido de nube pública-nube privada, como en el uso de servicios en nube heterogéneos.

3.1.7 Escalable Masivo

Definida como aquella estructura con un gran despliegue de sistemas o como una estructura simple pero diseñada para soportar una gran carga de trabajo. Son de reseñar las consideraciones de sobrecarga de proceso producidas al crecer el número de componentes de cualquier cloud de propósito general que evolucione de forma natural a un sistema de este tipo.

3.2 Selección de Caso de Uso

Atendiendo a los parámetros de desarrollo del presente trabajo, parece obvio que el sistema que mejor representa el despliegue deseado de todos los presentados en el punto anterior es el de sistema orientado a computación, cuyas características más relevantes para nuestro caso de estudio detallaremos a continuación:

- **Rendimiento:** Entendido como la capacidad y tiempo destinados por el sistema a la ejecución de aplicaciones de usuario final.
- **Almacenamiento:** Capacidad de persistencia, organización y disponibilidad de los datos de usuario final.
- **Gestión:** Facilidad de despliegue, configuración, alta y baja de nodos en un clúster.
- **Escalabilidad:** Capacidad del sistema para adaptarse dinámicamente a cargas variables de trabajo.

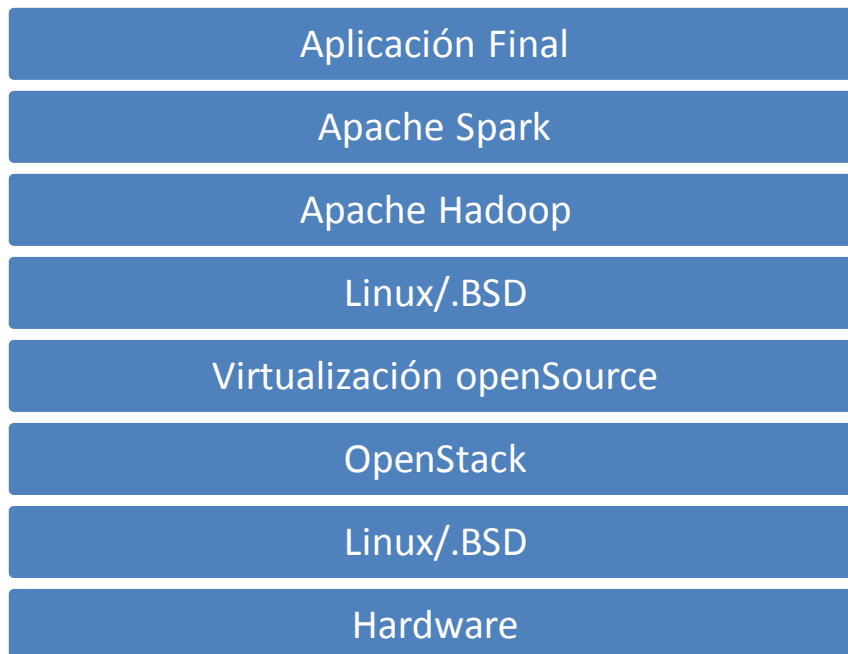
Es por tanto dicho uso final sobre el que desplegaremos nuestra pila de procesos, buscando una arquitectura adecuada para cumplir con los requisitos arriba detallados.

3.3 Estructuras Propuestas

Atendiendo a los puntos arriba detallados, podemos concluir las siguientes estructuras óptimas desde el punto de vista arquitectónico, aportando funcionalidades y facilidades de gestión y despliegue con complejidad creciente a la vez que atendiendo al rendimiento del sistema de usuario final.

3.3.1 Estructura Clásica

Corresponde a la arquitectura presentada en la introducción del presente trabajo, y se repite aquí para facilidad de consulta.



Aprovecharemos aquí para detallar los problemas inherentes a este tipo de despliegues:

- **Redundancia de almacenamiento:** Se emplean por separado para funciones disjuntas el almacenamiento nativo de la máquina anfitrión, los posibles sistemas de almacenamiento de objetos o sistema de ficheros de OpenStack, el sistema de ficheros virtualizado de las máquinas hospedadas y el sistema HDFS del entorno Hadoop.
- **Redundancia de librerías:** Se despliegan dos sistemas operativos de forma concurrente, cada uno con sus librerías y software mínimos duplicados tanto en la pila de procesos como en memoria.
- **Incremento de tamaño del sistema:** La transferencia de un sistema a otro nodo pasa por el portado de la máquina virtual completa, incluyendo los entornos Hadoop y Spark subyacentes.
- **Penalización en rendimiento:** El uso de máquinas virtuales incurrirá en una penalización de acceso a las funciones de entrada/salida, así como al acceso a memoria y tiempo de procesador particionados, así como en el coste de la traducción de funciones privilegiadas del procesador.

3.3.2 Estructura Mínima

Esta estructura centra la gestión y distribución de procesos en el sistema apache Spark para cada nodo. No ofrece facilidades de despliegue y/o configuración nodal, pero asegura un rendimiento óptimo del hardware subyacente.

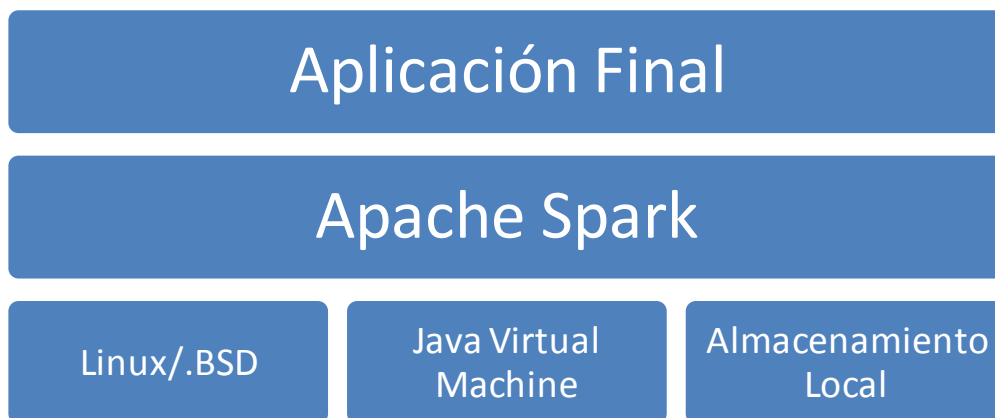


Ilustración 15 - Estructura Mínima

3.3.3 Estructura Gestionada

Esta estructura incluye un gestor de despliegue y configuración para los nodos, pero emplea virtualización ligera basada en contenedores para el despliegue de los sistemas de ejecución de procesos, así como de la gestión de almacenamiento distribuido. Por ello puede considerarse una arquitectura equilibrada en cuanto a gestión de recursos, replicabilidad y eficiencia en el uso de los mismos.

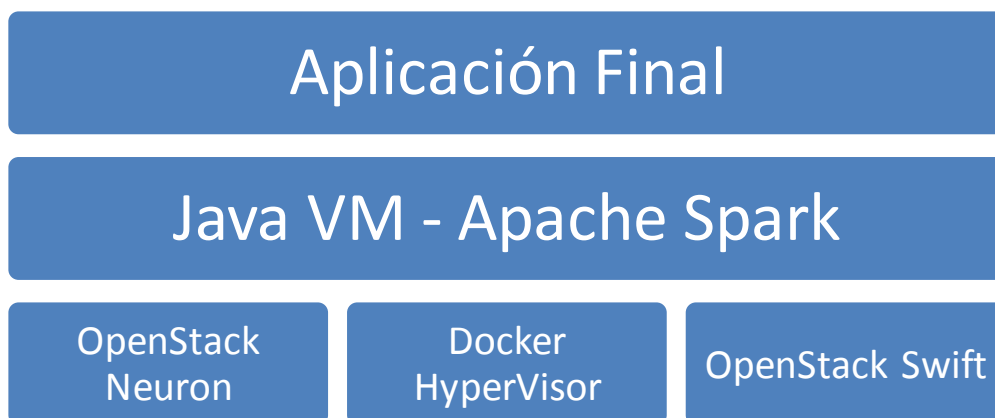


Ilustración 16 - Estructura Gestionada

4 Conclusiones

Donde una vez conocido el entorno tecnológico actual y los requisitos a cumplir por nuestro sistema, procederemos a realizar una comparativa de la cobertura de los mismos, para posteriormente proponer una arquitectura óptima. Finalmente realizaremos una valoración lo más objetiva posible de los resultados obtenidos, e indicaremos futuros progresos posibles en el entorno de estudio.

4.1 Cobertura de Objetivos

Sobre los objetivos fijados en el apartado 3.1 y desarrollados en el apartado 3.2, realizaremos un análisis de la cobertura de los mismos por las dos arquitecturas propuestas.

4.1.1 Estructura Mínima

Analizaremos a continuación la adecuación a nuestros objetivos de esta estructura, citada en el punto 3.3.1.

- **Rendimiento:** La eliminación de cualquier tipo de entorno de gestión aproxima el sistema de usuario lo más posible al entorno barebones, garantizando la mayor eficiencia posible en ejecución de funciones de usuario.
- **Almacenamiento:** El almacenamiento empleado es el propio de la máquina hospedadora, disponiendo de las características ofrecidas de forma autónoma por el mismo.
- **Gestión:** No se ofrecen facilidades de gestión. El despliegue y configuración de nuevas máquinas ha de desarrollarse manualmente. La tolerancia a fallos es la ofrecida por el entorno nativo.
- **Escalabilidad:** Un número elevado de máquinas hace inviable la gestión y administración de las mismas.

Es inmediato comprobar que esta estructura no cumple los requisitos de almacenamiento, gestión y escalabilidad arriba citados, por lo que cesará aquí su consideración.

4.1.2 Estructura Gestionada

- **Rendimiento:** El empleo de contenedores abre las puertas al uso de un solo núcleo monolítico por máquina, con acceso directo de cada contenedor a los recursos de entrada y salida de la misma, así como al núcleo y memoria de la misma. El rendimiento por tanto será cercano al de la máquina hospedadora,

penalizado tan solo por la traducción de espacios de direcciones para los procesos anidados.

- **Almacenamiento:** Es posible la utilización de sistemas de ficheros tradicionales vía OpenStack Cinder o el uso de redundancia a nivel de objetos con Swift.
- **Gestión:** La gestión de máquinas puede realizarse por medio del Dashboard OpenStack, pudiendo ser éstas levantadas, dadas de baja o reconfiguradas dinámicamente.
- **Escalabilidad:** La facilidad de gestión de recursos hardware arriba citada junto con la estructura de distribución de procesos entre nodos de Spark indica la posibilidad de adecuar el rendimiento de los nodos de computación a la carga dinámica de los mismos, por lo que Spark garantiza la escalabilidad de esta solución.

4.2 Arquitectura Óptima

Atendiendo al análisis arriba realizado, podemos postular la arquitectura gestionada como la óptima en adecuación a unas necesidades de explotación preestablecidas.

Resulta dudosa la posibilidad de mejorar el rendimiento de la misma sin renunciar a facilidades ya identificadas y definidas como necesarias para garantizar el correcto funcionamiento de un clúster determinado.

Cualquier reducción de complejidad posterior pasaría por la integración de funcionalidades de los sistemas OpenStack-Spark, como se mostrará en el punto siguiente.

4.3 Conclusiones

Lejos de ser un sistema cerrado, el campo de la computación distribuida es actualmente un entorno en constante cambio, en el que abundan las posibilidades de modularización y las alternativas de implementación para todos los componentes del entorno.

En el presente trabajo se espera haber capturado al menos parcialmente los conceptos básicos que dominan los despliegues en dicho entorno, así como las selecciones más comunes en el despliegue de los componentes arriba citados, justificando las propiedades y características más relevantes de las mismas.

Por último, se ha intentado la búsqueda de equilibrio entre costes de despliegue y mantenimiento y trabajo útil realizado por los sistemas, esquematizando el comportamiento de los mismos y seleccionando las funciones de interés y relevancia pertinentes.

4.4 Vías de Investigación Futuras

Como ya se ha señalado arriba, cualquier mejora previsible en la arquitectura diseñada pasaría por la integración o modularización de sistemas para un solo entorno. Se detallan aquí varios puntos de interés descritos como posible mejora en la definición de las arquitecturas específicas estudiadas:

- Implementación de un sistema de ficheros distribuido en Apache Spark: Eliminaría la redundancia en la que incurrimos al desplegar sobre Hadoop.
- Implementación de un sistema de gestión de nodos en Apache Spark. Eliminaría la necesidad de uso de Hadoop u OpenStack para el despliegue de nodos “ad-hoc”.
- Extensión de OpenStack Sahara para el soporte nativo de Apache Spark: De nuevo eliminaría la redundancia de funcionalidades de Hadoop.

Bibliografía

- [1] «Apache Spark™ - Lightning-Fast Cluster Computing». [En línea]. Disponible en: <https://spark.apache.org/>. [Accedido: 04-mar-2015].
- [2] «Spark Overview - Spark 1.2.1 Documentation». [En línea]. Disponible en: <https://spark.apache.org/docs/latest/>. [Accedido: 04-mar-2015].
- [3] «Spark SQL and DataFrames - Spark 1.3.0 Documentation». [En línea]. Disponible en: <https://spark.apache.org/docs/latest/sql-programming-guide.html>. [Accedido: 26-mar-2015].
- [4] «Spark Streaming - Spark 1.3.0 Documentation». [En línea]. Disponible en: <https://spark.apache.org/docs/latest/streaming-programming-guide.html>. [Accedido: 26-mar-2015].
- [5] «MLlib - Spark 1.3.0 Documentation». [En línea]. Disponible en: <https://spark.apache.org/docs/latest/mllib-guide.html>. [Accedido: 26-mar-2015].
- [6] «GraphX - Spark 1.3.0 Documentation». [En línea]. Disponible en: <https://spark.apache.org/docs/latest/graphx-programming-guide.html>. [Accedido: 26-mar-2015].
- [7] «Cluster Mode Overview - Spark 1.3.0 Documentation». [En línea]. Disponible en: <https://spark.apache.org/docs/latest/cluster-overview.html>. [Accedido: 26-mar-2015].
- [8] «1. Apache Hadoop core components - Getting Started Guide». [En línea]. Disponible en: http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2_1.html. [Accedido: 13-abr-2015].
- [9] «Welcome to Sahara! – Sahara». [En línea]. Disponible en: <http://docs.openstack.org/developer/sahara/>. [Accedido: 13-mar-2015].
- [10] «Rationale – Sahara». [En línea]. Disponible en: <http://docs.openstack.org/developer/sahara/overview.html>. [Accedido: 13-mar-2015].
- [11] «OpenStack Hadoop - Sahara». [En línea]. Disponible en: <https://software.mirantis.com/key-related-openstack-projects/savanna-openstack-hadoop/>. [Accedido: 13-mar-2015].
- [12] «Sahara REST API v1.0 – Sahara». [En línea]. Disponible en: http://docs.openstack.org/developer/sahara/restapi/rest_api_v1.0.html. [Accedido: 13-mar-2015].
- [13] «The Xen Project, the powerful open source industry standard for virtualization.» [En línea]. Disponible en: <http://www.xenproject.org/>. [Accedido: 26-mar-2015].
- [14] «Oracle VM VirtualBox». [En línea]. Disponible en: <https://www.virtualbox.org/>. [Accedido: 26-mar-2015].
- [15] «Main Page - KVM». [En línea]. Disponible en: http://www.linux-kvm.org/page/Main_Page. [Accedido: 26-mar-2015].
- [16] «QEMU». [En línea]. Disponible en: http://wiki.qemu.org/Main_Page. [Accedido: 26-mar-2015].
- [17] «Linux Containers». [En línea]. Disponible en: <https://linuxcontainers.org/>. [Accedido: 26-mar-2015].
- [18] «OpenVZ Linux Containers Wiki». [En línea]. Disponible en: https://openvz.org/Main_Page. [Accedido: 26-mar-2015].

- [19]«OpenStack - Wikipedia, the free encyclopedia». [En línea]. Disponible en: <http://en.wikipedia.org/wiki/OpenStack>. [Accedido: 13-mar-2015].
- [20]«OpenStack». [En línea]. Disponible en: https://wiki.openstack.org/wiki/Main_Page. [Accedido: 13-mar-2015].
- [21]«OpenStack Installation Guide for Debian 7 - juno». [En línea]. Disponible en: <http://docs.openstack.org/juno/install-guide/install/apt-debian/content/index.html>. [Accedido: 13-mar-2015].
- [22]«OpenStack Architecture Design Guide - current». [En línea]. Disponible en: <http://docs.openstack.org/arch-design/content/>. [Accedido: 13-mar-2015].