UNIVERSIDAD DEL BIO-BIO

FACULTAD DE CIENCIAS EMPRESARIALES

DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN

UNIVERSIDAD DEL BÍO-BÍO

"Estudio de factibilidad de almacenamiento de datos distribuido basado en base de datos no estructuradas para un sistema de monitoreo de alto volumen"

Proyecto de Software Aplicado presentado en conformidad a los requisitos para obtener el título de Ingeniero de Ejecución en Computación e Informática.

Alumno:
Leonel Ignacio Peña Luco

Profesor Guía:
Sr. Juan Carlos Parra Márquez

Concepción, Marzo de 2013

Agradecimientos

En primer lugar, agradecer a mi familia, a mi mamá por tener siempre la fortaleza de seguir adelante sin importar lo que suceda, a mi papá por haberme dado la vida, a mi hermano por ser como es y a mis hermanas por hacer la vida más dulce.

A Celeste por su incondicional apoyo.

Al profesor Patricio Gálvez por guiarme desde el inicio de mis estudios universitarios.

A Tzu Shen por sus exhaustivas revisiones, buena voluntad para responder todas mis dudas y por haber hecho posible este trabajo.

Y a todos los del departamento de computación de ALMA, que de alguna u otra forma contribuyeron en el desarrollo exitoso del mismo. En especial a Soledad y Álvaro.

Muchas Gracias.

Leonel Ignacio Peña Luco.

Resumen

La investigación realizada en este estudio de factibilidad, es parte del proyecto Solución de Almacenamiento para el TMC, cuyo objetivo es mejorar el actual sistema de almacenamiento de los datos monitoreados de las antenas. Este trabajo contempla el diseño de soluciones que mejoren el rendimiento del motor de almacenamiento persistente.

Esta tesis, muestra las ventajas de diseñar esquemas que obtengan provecho de las características de la base de datos utilizada. Dichos esquemas son analizados, diseñados y evaluados mediante la ejecución de pruebas de rendimiento, para luego analizar los resultados obtenidos del diseño propuesto y compararlos con el diseño actual.

Finalmente, se evalúa la factibilidad de implantar en producción el diseño propuesto.

Abstract

The research carried out in this feasibility study, is part of the project storage solution for the TMC, which objective is to improve the storage of the monitored data of the antennas. This work involves the design of solutions that improve the performance of persistent storage engine. This thesis, shows the advantages of designing schemes that obtain profit of the characteristics of the used database.

Such schemes are analyzed, designed and evaluated through the implementation of performance tests, and then analyze the results of the proposed design and compare them with the current design. Finally, it assesses the feasibility of deploying in production the proposed design.

Índice de contenido

Índice de ilustraciones

Índice de Tablas

1 INTRODUCCIÓN

La gestión de grandes volúmenes de información es un tema comúnmente discutido en los últimos cinco años y no es para menos, la era digital que se está viviendo ha empujado a varias empresas tecnológicas como Google, Facebook, Twitter y Amazon a buscar nuevas formas y conceptos de organización y tratamiento de datos. Tanto es así, que muchos de ellos han creado sus propios motores de almacenamiento forjados bajo conceptos distintos a los tradicionales.

Dichos motores de almacenamiento se engloban en un conjunto de tecnologías denominado NoSQL, que están directamente relacionadas con el procesamiento de grandes volúmenes de datos, paralelización de procesos, alta escalabilidad y disponibilidad, con soluciones de calidad y a bajos costos, muchas de ellas OpenSource.

ALMA es un observatorio astronómico compuesto por antenas de alta precisión que, debido a su complejidad, requieren ser monitoreadas constantemente. Para el almacenamiento de los datos generados se utiliza MongoDB, que es una base de datos de la familia NoSQL.

Este trabajo de tesis estudia las mejoras que pueden ser implementadas para aumentar el rendimiento de MongoDB.

Para comenzar, se presenta una breve historia sobre las bases de datos NoSQL, sus características y clasificación. Luego, se realiza una guía sobre las funcionalidades de MongoDB, sus operaciones y comportamiento en distintos aspecto como el indexado y el sharding.

La correcta identificación y comprensión de un problema, es clave para la creación de soluciones de calidad, es por ello, que el capítulo de análisis abarca en detalle todos los aspectos necesarios para comprender las estructuras de datos involucradas en el almacenamiento de los datos del monitoreo. A partir de ello, se procede a estudiar minuciosamente el esquema utilizado actualmente para almacenar los datos del monitoreo, sus conceptos e impactos.

Identificada y comprendida la naturaleza del problema, se procede al diseño de soluciones que logren

solventar la problemática. Para ello, el capítulo de diseño propone tres esquemas para MongoDB, exhibiendo sus ventajas y desventajas, estimando el impacto de cada uno de ellos y, finalmente, presentando una comparación para apoyar la elección, de manera cuantitativa, del diseño más apto de implementar.

A manera de completar la documentación, se presenta la puesta en marcha de un servidor con MongoDB, cubriendo su instalación, configuración y uso.

Una vez que las soluciones ya se encuentran diseñadas, se continúa con la etapa de pruebas. Para ello, el capítulo de pruebas realiza todas las evaluaciones necesarias para determinar si el diseño seleccionado cumple o no los requerimientos del TMC.

Finalmente, se realiza un estudio de factibilidad basado en el rendimiento del esquema actual y los resultados obtenidos en las pruebas del esquema propuesto.

2 ATACAMA LARGE MILLIMETER/SUBMILLIMETER ARRAY

2.1 ¿Qué es ALMA?

ALMA, Atacama Large Millimeter/Submillimeter Array, es una instalación astronómica internacional construida en asociación entre Europa, Norteamérica y Asia del Este en colaboración con la República de Chile, es el mayor proyecto astronómico del mundo.

Con un diseño revolucionario, ALMA está compuesto, inicialmente, por 66 antenas de alta precisión que trabajan conjuntas como un único telescopio. Esto permitirá a ALMA observar el Universo frío, desde el gas molecular y el polvo, hasta los vestigios de la radiación del Big Bang. Estudiará los componentes básicos de las estrellas, los sistemas planetarios, las galaxias y la vida misma. Proporcionará a los científicos imágenes detalladas de estrellas y planetas naciendo en nubes de gas, y detectará galaxias distantes en formación en los límites del Universo observable, que se ven tal y como eran hace unos diez mil millones de años.

El término de la construcción de ALMA está programada para Marzo del año 2013.

2.2 Área de Estudio

El departamento de computación de ALMA, ADC (ALMA Department of Computing) es la unidad encargada de la infraestructura informática de ALMA.

ADC está compuesto por 3 equipos:

- Software
- Tecnologías de Información
- Grupo de Operadores de Archivo

Este estudio se desenvuelve en el grupo de software.

A continuación se presenta el organigrama del departamento.

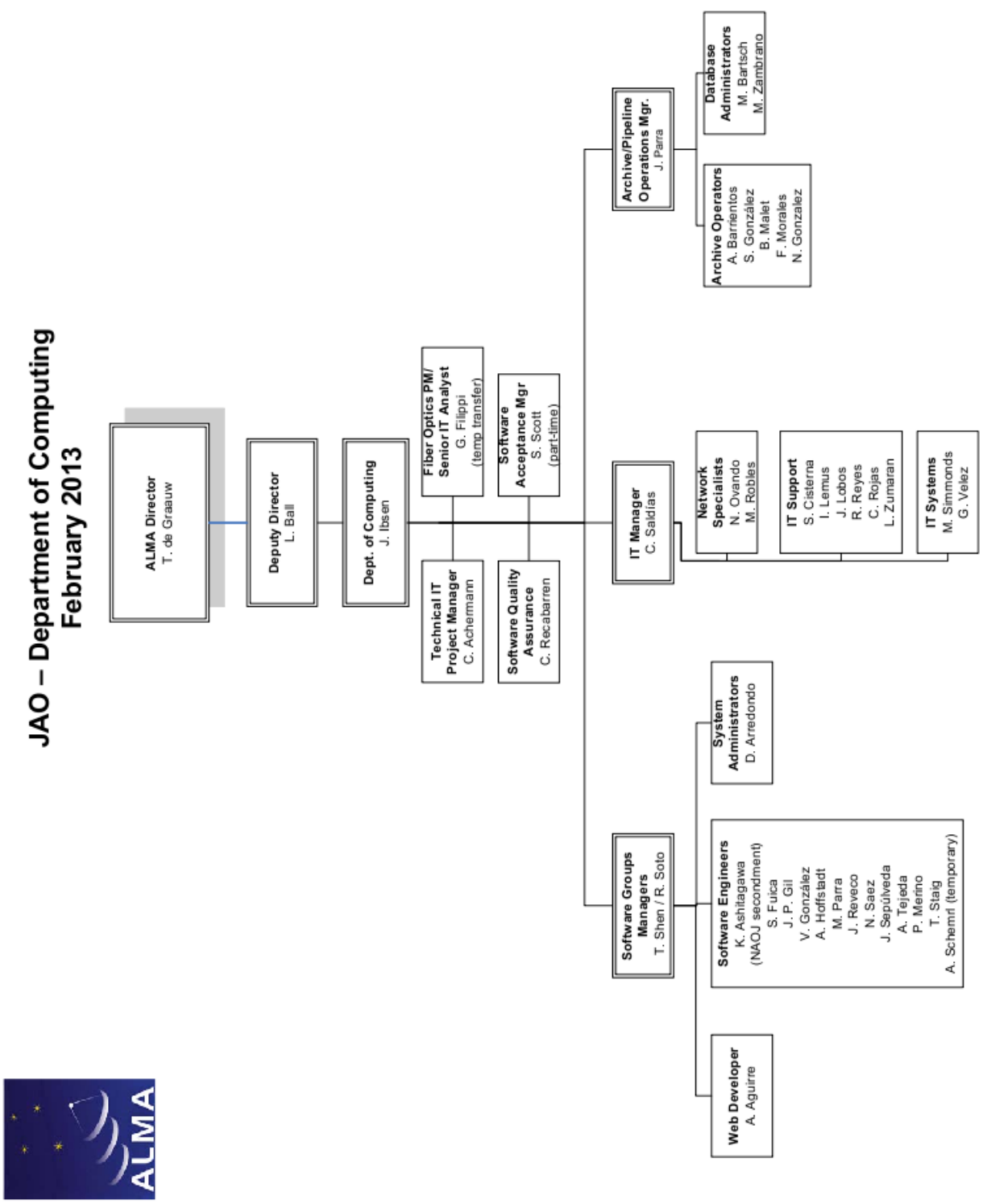


Ilustración 1: Organigrama de ADC

2.3 Descripción del Problema

Debido a la gran complejidad que significa mantener todas las antenas funcionando sincronizadamente y en perfecto estado, ALMA posee un sistema de monitoreo, de entre muchos otros, denominado TMC, que es el encargado de gestionar los datos recolectados del monitoreo de las antenas.

El departamento de computación de ALMA está evaluando la implementación de MongoDB para el almacenamiento de los datos de monitoreo. Para ello ha dispuesto de un cluster en el cual se han insertado datos reales correspondientes al monitoreo. El flujo entrante de datos genera 4.000 inserciones por segundo. Finalizada la construcción de ALMA, se esperan (y en pleno funcionamiento del arreglo de antenas) 8.000 inserciones por segundo y 40GB de datos diariamente.

El gran flujo de datos es insertado sin problemas por MongoDB, no siendo así su recuperación. El tiempo requerido para ello es variado, pero en general toma entre 5, 10 o incluso 15 minutos recibir 3 horas de datos, lo cual es demasiado.

Esto es un problema crítico, ya que los científicos e ingenieros necesitan los datos del monitoreo para detectar y solucionar posibles problemas en el hardware, mal funcionamiento de las antenas o de los equipos centrales del arreglo, tales como: el correlacionador, referencias fotónicas, etc. Dichos datos deben ser recuperados en el menor tiempo posible para no entorpecer ni retrasar las tareas de detección y corrección de fallas.

Es por esto que ADC requiere de un estudio de factibilidad para determinar si MongoDB es capaz de entregar los resultados de las consultas en un período razonable de tiempo, independientemente de que la base de datos contenga varios cientos de millones de registros.

2.4 Telescope Monitoring Control

2.4.1 ¿Qué es el TMC?

El TMC, Telescope Monitoring and Control, es un sistema compuesto por una variedad de aplicaciones que en conjunto gestionan los datos provenientes del monitoreo de las antenas.

Las funciones que realiza el TMC son: procesar los mensajes entrantes desde ACS, publicar los datos monitoreados, almacenar permanentemente las muestras y generar ficheros de texto con los datos monitoreados para su exposición a través de una interfaz web.

2.4.2 Requerimientos

Las condiciones bajo las cuales debe funcionar el TMC están directamente relacionadas con la cantidad de puntos de monitoreo y su tiempo de muestreo. Mientras mayor sea el número de puntos monitoreados y su tiempo de muestreo sea menor, mayor serán los requerimientos computacionales (procesamiento, almacenamiento físico, memoria, etc.) del TMC.

Los requerimientos para el almacenamiento permanente son:

1. Almacenar los datos de 100.000 variables.
2. Procesar 8.000 inserciones por segundo.
3. Tiempo de extracción requerido para 1 variable con datos de 24 horas tiene que ser menor a 1 segundo.

2.4.3 Infraestructura

La infraestructura del TMC ya se encuentra implementada y está compuesta desde el punto de vista del software, por un conjunto de aplicaciones que se encargan de distintas funcionalidades, tales como: el almacenamiento permanente, el almacenamiento volátil, la gestión de mensajes, entre otros.

Esta aplicaciones son las siguientes:

- Apache ActiveMQ: Cola que mantiene los mensajes entrantes enviados desde ACS a la espera de ser consumidos.
- Redis: Base de datos en memoria que se encarga de publicar los datos del monitoreo y del almacenamiento temporal de los datos más recientes.
- MongoDB: Base de datos NoSQL que se encarga del almacenamiento permanentemente de los datos.
- TMC Offline: Aplicación escrita en el lenguaje Java y que consume los mensajes desde ActiveMQ, publicándolos en Redis, enviándolos a MongoDB y generando los ficheros de texto para Monitor Web Dashboard.
- Monitor Web Dashboard: Aplicación web escrita en el lenguaje Python que presenta los datos a los científicos e ingenieros.

La finalidad de este estudio es mejorar el rendimiento de MongoDB, es por ello que el

comportamiento y aspectos detrás de Apache ActiveMQ y las demás aplicaciones que no están relacionadas directamente con él, son explicadas a grandes rasgos, ya que solamente es necesario saber lo que hacen y no afectan, de ningún modo, el desarrollo de este estudio.

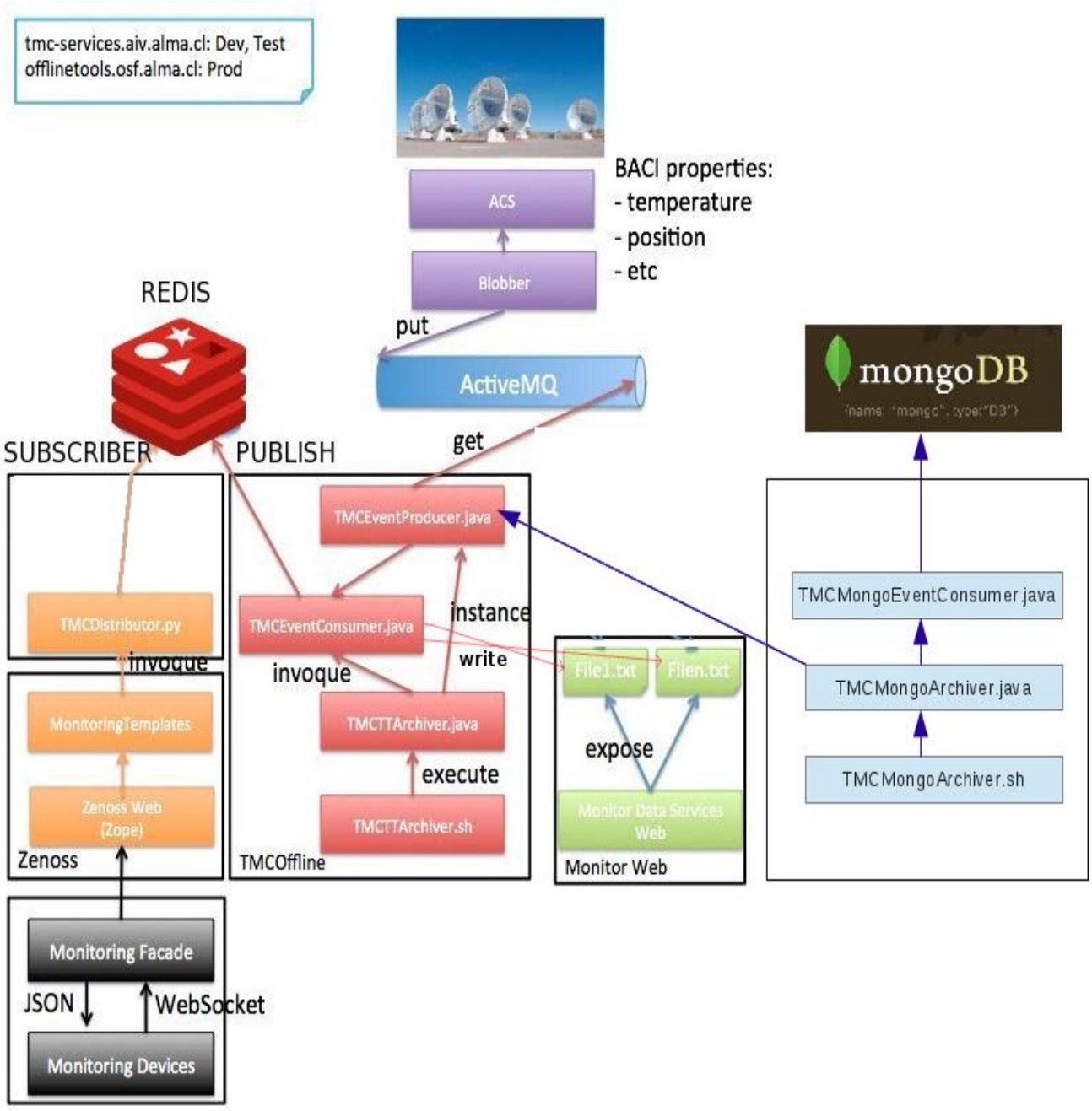


Ilustración 2: Arquitectura del TMC

3 DEFINICIÓN DEL PROYECTO

3.1 Objetivos

3.1.1 Objetivo General

Implementar y evaluar el desempeño de las bases de datos no relacionales para minimizar los tiempos de consulta de los datos de monitoreo de las antenas.

3.1.2 Objetivos Específicos

1. Analizar la arquitectura actual del manejo de los datos de monitoreo.
2. Diseñar e implementar experimentos para evaluar el desempeño de las alternativas que ofrece el mercado versus la solución actual, utilizando los datos de monitoreo reales.
3. Diseñar e implementar esquemas que se ajusten a los datos de monitoreo en pos de lograr un rendimiento eficiente tanto en la inserción como también en la extracción.
4. Evaluar el desempeño de las bases de datos bajo los criterios de número de inserciones por segundo y tiempo de retorno en la obtención de datos.

3.2 Metodología

La metodología que utilizará este estudio de factibilidad está basado en el método ingenieril, pero con algunas modificaciones.

Este método consiste en las siguientes etapas:

1. Identificación del problema
2. Recopilación de información
3. Búsqueda de soluciones
4. Diseño de experimentos
5. Evaluación de las soluciones
6. Elaboración y presentación de los resultados
7. Documentación

3.3 Definiciones, Siglas y Abreviaciones

.deb	:	Extensión utilizada por el sistema operativo Debian y derivados para identificar sus paquetes.
10gen	:	Empresa detrás del desarrollo de MongoDB.
ACS	:	ALMA Common Software.
ActiveMQ	:	Cola de mensajes de alto rendimiento y disponibilidad escrita en Java.
ALMA	:	Atacama Large Millimeter/Submillimeter Array.
ALMASW	:	Sistema de Control de ALMA.
AOS	:	Sitio de operaciones del conjunto de antenas.
API	:	Interfaz de programación de aplicaciones, es un conjunto de funciones y procedimientos que ofrece un software para ser utilizado como una capa de abstracción para otro software.
APT	:	Herramienta que gestiona los paquetes en debian.
Blobber	:	Estructura que recolecta, encapsula y envía los datos del monitoreo de las antenas.
BSON	:	Notación Binaria de Objetos (Binary JSON), es una versión modifica de JSON que permite búsquedas rápidas de datos.
Cardinalidad	:	Número de chunks de una llave compartida.
Chunk	:	Es una porción de datos de una colección compartida que representa a una pequeña parte del rango de llaves.
Cluster	:	Conjunto de máquinas que se comportan como si fuesen una única computadora.
Debian	:	Sistema operativo de código abierto basado en el kernel Linux.
Epoch	:	Sistema de medición del tiempo en sistemas operativos Unix y contiene el número de segundos transcurridos desde 1970.
Framework	:	Es una infraestructura digital que cuenta con módulos de software, que sirven como base para que otro proyecto de software puede ser más fácilmente organizado y desarrollado.
GB	:	Gigabytes.
GridFS	:	Funcionalidad que permite a MongoDB almacenar ficheros de gran tamaño (más de 16 MB) en una base de datos.
Javascript	:	Lenguaje de programación interpretado.
JSON	:	JavaScript Object Notation, es una notación para representar un objeto.
KB	:	Kilobytes.
Mainframe	:	Computador central utilizado para el procesamiento de gran cantidad de datos.
MapReduce	:	Framework para el procesamiento paralelo y distribuido de datos.
MB	:	Megabytes.

Monitor Web Dashboard	:	Aplicación web para la visualización de los datos de monitoreo.
NoSQL	:	No sólo SQL.
OpenSource	:	Software distribuido y desarrollado gratuitamente.
OSF	:	Centro de operaciones de ALMA.
Paquete	:	Formato de distribución de programas en Gnu/Linux.
RDBMS	:	Sistema administrador de bases de datos relacional.
Redis	:	Motor de base de datos en memoria, basado en el almacenamiento clave-valor.
SCO	:	Sede central de ALMA en Santiago de Chile.
Tabla Hash	:	Estructura de datos que asocia claves con valores.
TB	:	Terabytes.
TMC	:	Telescope Monitoring and Control.
TMC Offline	:	Aplicación escrita en java que gestiona los mensajes consumidos desde ActiveMQ.
TMCMongo Archiver	:	Clase escrita en Java que se encarga de insertar en MongoDB los datos del monitoreo de las antenas.
Web 2.0	:	Aplicaciones que generan colaboración y servicios que reemplazan las aplicaciones de escritorio por las de Internet.

4 MARCO TEÓRICO

4.1 NoSQL

4.1.1 ¿Qué es NoSQL?

NoSQl o "Not Only SQL" es un término utilizado para denotar a todas las bases de datos y almacenes de datos que no siguen los principios del modelo relacional, que a menudo se relacionan con grandes conjuntos de datos accedidos y manipulados a escala Web[1].

NoSQL no es un solo producto o una tecnología. Es una clase de productos y una colección de diversos, y a veces relacionados, conceptos sobre almacenamiento y manipulación de datos a gran escala.

4.1.2 Breve Historia

Las bases de datos no relacionales no son nuevas, de hecho fueron los primeros sistemas de almacenamiento que se utilizaron al tiempo en que las primeras máquinas de computación fueron inventadas. Las bases de datos no relacionales prosperaron con la llegada de los mainframes y han existido en ámbitos específicos y especializados[1].

Con la llegada de los primeros motores de almacenamiento relaciones en los años 70s, los sistemas no relaciones fueron desplazados rápidamente, ya que presentaban ciertas ventajas que se potenciaban con el entorno que se tenía en aquel entonces, por ejemplo: poca escalabilidad, datos uniformes, aplicaciones Web básicas, etc. Pero con el advenimiento de la Web 2.0[2], muchos de las bases de datos relacionales se vieron sobrepasadas en su rendimiento debido a la paralelización, la escalabilidad masiva, manejo de gigantescos volúmenes de datos y el costo de mantención.

Los sistemas NoSQL tienen su renacer en el mundo bajo el alero de las aplicaciones de Internet de escalabilidad masiva, computación paralela y sistemas distribuidos. Gran parte de este nuevo enfoque se debe a las soluciones que publicó Google a través de papers[3].

En Google construyeron una infraestructura escalable para el procesamiento paralelo de grandes cantidades de datos. Ellos crearon una solución para cada una de los problemas presentes en la pila

1 Red de alcance mundial que conecta a cientos de millones de equipos para usos de los más variados, tales como: doméstico, empresarial, científico, oseo, entre otros.

de aplicaciones. Un sistema de ficheros distribuido[4], un almacén de datos orientado a familia de columnas[3], un sistema de coordinación distribuido[5] y un entorno de ejecución basado en el algoritmo paralelo MapReduce[6].

Transcurridos unos pocos años desde las publicaciones de Google, muchas compañías han adoptado los conceptos del NoSQL para la gestión de grandes datos, Facebook, Netflix, Yahoo, EBay, Hulu, IBM son algunas de ellas[1].

4.1.3 Clasificación

No todas las bases de datos NoSQL son similares, ni están hechas para resolver los mismos problemas. La Tabla 1 describe las categorías más populares que presentan estos sistemas y los clasifica de acuerdo a la forma en que almacenan los datos.

Categoría	Descripción	Nombres de bases de datos
Documentales	Los datos son almacenados como documentos JSON o sus derivados.	MongoDB, RavenDB, CouchDB, IBM Lotus, Jackrabbit, OrientDB, SimpleDB, Terrastore
XML	Los datos son almacenados en formato XML.	BaseX, eXist, MarkLogic Server
Grafos	Los datos son almacenados como una colección de nodos, donde los nodos son análogos a objetos en los lenguajes de programación. Los Nodos son conectados usando aristas de relaciones.	AllegroGraph, DEX, Neo4j, FlockDB, Sone GraphDB, HyperGraphDB, InfoGrid, InfiniteGraph
Clave-valor	En una base de datos clave-valor, los valores son almacenados como cadenas. Parecidas a las estructura hash.	Cassandra, Dynamo, Hibari, Riak, Redis, BigTable, HBase, Hypertable
Objeto	Los datos son representados como objetos, los mismo que se utilizan en los lenguajes de programación orientados a objetos.	Zope Object Database, db4o, GemStone S, Objectivity/Db

Tabla 1: Clasificación bases de datos NoSQL

4.1.4 Características

4.1.4.1 Consistencia eventual

No se implementan mecanismos rígidos de consistencia como los presentes en las bases de datos relacionales, donde la confirmación de un cambio implica una comunicación del mismo a todos los nodos que lo repliquen. Esta flexibilidad hace que la consistencia llegue, eventualmente, cuando no

se han ocurrido modificaciones en los datos durante un periodo de tiempo[7].

4.1.4.2 Estructura distribuida

Se refiere a la distribución de los datos mediante mecanismos de particionado, los cuales reparten porciones de la base de datos a través del cluster completo.

4.1.4.3 Escalabilidad horizontal

La escalabilidad horizontal significa añadir más nodos a un sistema, los cuales proveerán de mayor capacidad de procesamiento y almacenamiento. En contraposición a la escalabilidad vertical, la cual consiste en máquinas con alto poder de procesamiento y almacenamiento, muchas de ellas soluciones de alto costo y privativas.

La gran mayoría de los sistemas NoSQL fueron diseñados para escalar horizontalmente en computadores comunes y corrientes. Esta escalabilidad horizontal otorga distribución de la carga a través de los nodos del cluster, permitiendo que los datos sean accesibles a mayor velocidad.

En los sistemas NoSQL, los datos almacenados pueden ser muchos más rápidos al tomar ventaja de la escalabilidad horizontal, lo que se traduce en la distribución de la carga a través de aquellos nodos.[1]

4.1.4.4 Tolerancia a fallos y redundancia

Los sistemas NoSQL están pensados para ser ejecutados en cientos o incluso miles de nodos, otorgando redundancia de datos y gran tolerancia a fallos. Los principales medios para lograr esto son la replicación y el particionado de datos mediante la estructura distribuida.

4.2 MongoDB

Este capítulo cubre todos los aspectos técnicos de MongoDB que este proyecto requiere para ser implementado, entregando una guía de referencia sobre las funcionalidades básicas de manipulación de registros: crear, actualizar, leer y remover.

A lo largo de todo el capítulo se utiliza un sólo ejemplo que es modificado en cada uno de los tópicos tratados, ejemplificando la evolución de un documento desde lo más básico hasta uno más complejo que es particionado entre los nodos de un cluster.

Además, se han cubierto en detalle dos temas esenciales: el uso y desempeño de índices y de

colecciones compartidas.

4.2.1 ¿Qué es MongoDB?

MongoDB es un sistema de base de datos NoSQL orientado a documentos. No requiere la creación de esquemas, es escalable y de alto rendimiento[8].

4.2.2 Conceptos básicos

MongoDB introduce una serie de nuevos conceptos que son necesarios comprender antes de adentrarse en el uso mismo de la base de datos.

La unidad básica de datos en MongoDB son los *documentos*, similares a las filas de las bases de datos relacionales. Estos *documentos* son organizados en *colecciones* (tablas en el modelo relacional) que a su vez pertenecen a una bases de datos. De esta manera una base de datos puede almacenar múltiples colecciones, las que a su vez guardan un sin fin de documentos.

4.2.2.1 Documentos

Los documentos son almacenados en formato BSON, que es una versión mejorada del formato JSON, y almacena los datos como un conjunto de pares clave/valor.

```
{
    nombre: 'Leonel'
}
```

Ilustración 3: Ejemplo de documento con un atributo

Debido a que el formato JSON es una representación de objetos, los documentos también representan a un objeto. Es así como el concepto "columna" de una fila se transforma en atributo de un documento.

```
{
    nombre: 'Leonel',
    edad:   25
}
```

Ilustración 4: Ejemplo de documento con dos atributos de distinto tipo

Al igual que todo objeto, los documentos también permiten la anidación de objetos dentro de un atributo. Aquellos documentos reciben el nombre de intra-documentos, subdocumentos o anidados. Para acceder a un objeto de este tipo se utiliza el carácter punto. Por ejemplo, para acceder al atributo edad en la Ilustración 5 se utiliza la notación "*ingeniero.edad".*

```
{
    ingeniero: {
        nombre: 'Leonel',
        apellido: 'Luco',
        edad: 25
    }
}
```

Ilustración 5: Ejemplo de documento anidado

Todos los documentos poseen el atributo *_id,* cuyo valor es único entre todos los documentos de una colección. No es posible insertar, en una misma colección, un documento que tenga el mismo valor *_id* que otro. Esta funcionalidad es análoga a las llaves primarias de las bases de datos relacionales.

Especificar el atributo *_id* en un documento no es obligatorio. Esto queda reflejado en los ejemplos anteriores en donde no se ingresa valor alguno para *_id*, es más ni siquiera se menciona. En aquellos casos, MongoDB generará automáticamente un valor único en *_id* para el documento que será insertado.

```
{
    _id : ObjectId("51257148633e731e8f251a5a"),
    nombre : "Leonel",
    edad : 25
}
```

Ilustración 6: Ejemplo de documento con identificador generado

El tamaño máximo que puede tener un documento es de 16 Megabytes. Sin embargo, si se implementa la funcionalidad de GridFS no existe un límite teórico, aunque si queda acotado por los recursos físicos del cluster en el que es ejecutado MongoDB.

4.2.2.2 Colecciones

Las colecciones almacenan los documentos y son análogas a las tablas del modelo relacional. En ellas se insertan, leen, actualizan y eliminan documentos, se crean índices, se realizan operaciones de mantenimiento, entre otras funcionalidades.

No existe límite en el número de documentos que una colección puede almacenar. Sin embargo, esto se encuentra acotado por la capacidad física de almacenamiento que tenga la máquina o cluster en el que se aloja MongoDB.

El número máximo de colecciones que MongoDB puede manejar por base de datos, sin implementar la funcionalidad de GridFS, es 24.000.

4.2.2.3 Esquema flexible

Los documentos en MongoDB son de esquema flexible. Esto quiere decir que cada documento dentro de una colección puede tener cualquier atributo y tipo de dato, lo tengan o no los demás documentos.

Esta característica será retomada, profundizada y ejemplificada en la sección *4.2.3 Creación y modificación de documentos*.

El proceso de creación de las bases de datos y colecciones es realizado automáticamente por MongoDB en tiempo de inserción y no requiere de ningún comando ni operación. Lo único que se debe hacer es enviar los datos como si la base de datos y colección existieran, MongoDB hará el resto.

4.2.3 Creación y modificación de documentos

La creación de documentos dentro de una colección en MongoDB puede ser realizada a través de las operaciones *insert* o *upsert.* También es posible introducir documentos a una colección por medio de la importación de respaldos, pero esa funcionalidad no será cubierta por este estudio.

Los documentos que ya se encuentran insertados en MongoDB pueden ser modificados a través de las operaciones *update* y *upsert*.

4.2.3.1 Operación insert

La sintaxis de la operación *insert* es:

db.coleccion.insert({ <documento> })

Por ejemplo, para insertar el documento presentado en la Ilustración 5 dentro de la colección *ingenieros,* el código es el siguiente:

db.ingenieros.insert({ _id: 1, ingeniero: {nombre: 'Leonel', apellido: 'Luco', edad: 25 }})

A diferencia de la ilustración, aquí se ha especificado un valor para el atributo *_id*, pero cabe recordar que si no es así, MongoDB generará uno automáticamente y siempre mantendrá la propiedad de unicidad dentro de todos los documentos de la colección.

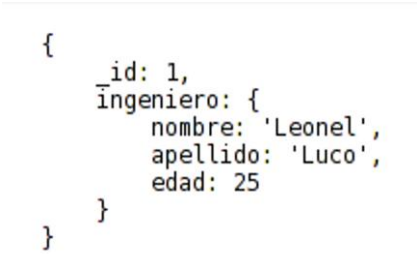


Ilustración 7: Ejemplo de documento con identificador especificado

También es posible insertar múltiples documentos por medio de una sola instrucción:

```
db.ingenieros.insert( [
        { _id: 2, nombre: 'Pedro', apellido: 'Correa' },
        { _id: 3, nombre: 'Juan', apellido: 'Castillo' },
        { _id: 4, nombre: 'Leonel', apellido: 'Sanchez' }
]);
```

4.2.3.2 Operación update

Todos los documentos almacenados en MongoDB pueden ser modificados por medio de la operación *update*. Su sintaxis es la siguiente:

```
db.colección.update( { <consulta> }, { <actualización> })
```

Por ejemplo, para modificar el valor del atributo *apellido* del documento { *_id: 1* }, la operación *update* es la siguiente:

```
db.ingenieros.update( { _id: 1}, { $set: {'ingeniero.apellido': 'Peña' }} )
```

La expresión *$set* debe preceder al objeto que será actualizado.

```
{
    _id: 1,
    ingeniero: {
        nombre: 'Leonel',
        apellido: 'Peña',
        edad: 25
    }
}
```

Ilustración 8: Documento resultante de actualizar el atributo apellido

La API de MongoDB ofrece la posible de agregar nuevos atributos a los documentos por medio de la expresión *$push*. Por ejemplo, para añadir una lista con los lenguajes de programación al documento *{ _id: 1}* se puede utilizar la siguiente instrucción:

```
db.ingenieros.update( { _id: 1}, { $push: {lenguajes: [ 'Java', 'PHP', 'C/C++' ]  }} )
```

Este ejemplo muestra el uso de esquemas flexibles, ya que el atributo *lenguajes* ha sido añadido en tiempo de actualización, dejando al documento *{ _id: 1 }* como el único que lo posee dentro de todos los demás documentos de la colección *ingenieros.*

```
{
    _id: 1,
    ingeniero: {
        nombre: 'Leonel',
        apellido: 'Peña',
        edad: 25
    },
    lenguajes: [ 'Java', 'PHP', 'C/C++' ]
}
```

Ilustración 9: Documento resultante de agregar el atributo lenguajes

También es posible eliminar atributos en tiempo de actualización. Para ello se utiliza la expresión $*unset.* Por ejemplo, para eliminar el atributo *edad* del documento *{_id: 1}* se utiliza la siguiente instrucción:

db.ingenieros.update({ _id: 1}, { $unset: {'ingeniero.edad': 1 }})

```
{
    _id: 1,
    ingeniero: {
        nombre: 'Leonel',
        apellido: 'Peña'
    },
    lenguajes: [ 'Java', 'PHP', 'C/C++' ]
}
```

Ilustración 10: Documento resultante de eliminar el atributo edad

La flexibilidad y potencia que ofrece la operación *update* posibilita la combinación de expresiones en una sola instrucción, lo que resulta más rápido que ejecutarlas individualmente.

La combinación de los ejemplos anteriores que operaban sobre los atributos *lenguajes* y *edad* en una misma instrucción es la siguiente:

```
db.ingenieros.update( { _id: 1}, {
        $push: {lenguajes: [ 'Java', 'PHP', 'C/C++' ] },
        $unset: {'ingeniero.edad': 1  }
})
```

En el mundo relacional, estas características de añadir y quitar atributos en tiempo de actualización no tiene equivalente, aunque puede ser imaginado como si a una tupla se le agregara una nueva columna, la cual no existe para las demás tuplas de la misma tabla.

4.2.3.3 Operación upsert

La operación *upsert* elimina la necesidad de consultar a la base de datos si es que un documento existe o no, con la intención de actualizarlo si es que está o insertarlo si es que no. La operación es la misma que una actualización, pero añadiendo un tercer parámetro: { *upsert: true }.*

La sintaxis es la siguiente:

```
db.colección.update( { <consulta> }, { <actualización> }, { upsert: true } )
```

El comportamiento de esta operación es el siguiente:

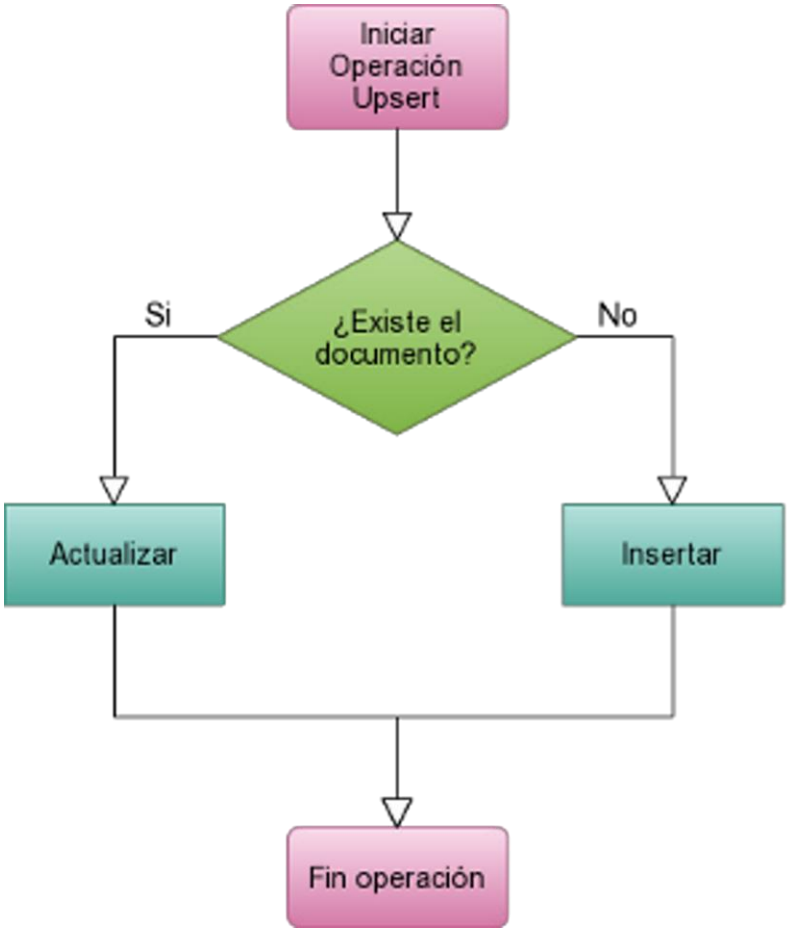


Ilustración 11: Diagrama de flujo de la operación Upsert

4.2.4 Lectura de documentos

La búsqueda de documentos dentro de una colección es realizada a través de los métodos *find()* y *findOne()*.

4.2.4.1 Método find()

El método *find()* busca todos los documentos que cumplan con <*condición*>, retornando un cursor con los elementos encontrados.

db.colección.find({ <condición> }, { <proyección> })

Por ejemplo, para obtener todos los documentos creados en la colección *ingenieros* se utiliza la siguiente instrucción:

db.ingenieros.find()

```
> db.ingenieros.find()
{ "_id" : ObjectId("51257148633e731e8f251a5a"), "nombre" : "Leonel", "edad" : 25 }
{ "_id" : 2, "nombre" : "Pedro", "apellido" : "Correa" }
{ "_id" : 3, "nombre" : "Juan", "apellido" : "Castillo" }
{ "_id" : 4, "nombre" : "Leonel", "apellido" : "Sanchez" }
{ "_id" : 1, "ingeniero" : { "apellido" : "Peña", "nombre" : "Leonel" }, "lenguajes" : [ [ "Java", "PHP", "C/C++" ] ] }
>
```

Ilustración 12: Listado de todos los documentos de la colección ingenieros

La Ilustración 12 muestra el estado final de todos los documentos de la colección *ingenieros*, cuyos atributos sufrieron cada uno de los cambios presentados en los ejemplos anteriores.

Se puede agregar el atributo *_id* como criterio de búsqueda:

db.ingenieros.find({ _id: 1 })

```
> db.ingenieros.find({ _id: 1 })
{ "_id" : 1, "ingeniero" : { "apellido" : "Peña", "nombre" : "Leonel" }, "lenguajes" : [ [ "Java", "PHP", "C/C++" ] ] }
>
```

Ilustración 13: Búsqueda de un documento a través de su identificador

Las consultas realizadas hasta el momento retornan el documento completo con todos sus atributos, en muchas situaciones esto no es necesario. En SQL, la instrucción SELECT contiene los nombres de las columnas que se desean seleccionar, proyectar o visualizar. En MongoDB, para obtener un determinado atributo o un grupo de ellos se utiliza el parámetro <proyección>. Existen dos valores que puede tomar un atributo proyectado: "0" para proyectar todo el documento **excepto** el atributo en cuestión y "1" para proyectar **solamente** el atributo involucrado. El atributo *_id* es incluido por defecto.

Por ejemplo, para obtener sólo el listado de lenguajes del documento *{ _id: 1 }* se utiliza la siguiente instrucción:

db.ingenieros.find({ _id: 1 }, { lenguajes: 1 })

y, para obtener todo el documento excepto los lenguajes:

db.ingenieros.find({ _id: 1 }, { lenguajes: 0 })

```
> db.ingenieros.find( { _id: 1 }, { lenguajes: 1 } )
{ "_id" : 1, "lenguajes" : [ [ "Java", "PHP", "C/C++" ] ] }
>
> db.ingenieros.find( { _id: 1 }, { lenguajes: 0 } )
{ "_id" : 1, "ingeniero" : { "apellido" : "Peña", "nombre" : "Leonel" } }
>
```

Ilustración 14: Ejemplo de proyección de atributos

Para obtener todos aquellos documentos en los que su atributo *nombre* es igual a *'Leonel':*

db.ingenieros.find({ nombre: 'Leonel' })

```
> db.ingenieros.find( { nombre: 'Leonel' } )
{ "_id" : ObjectId("51257148633e731e8f251a5a"), "nombre" : "Leonel", "edad" : 25 }
{ "_id" : 4, "nombre" : "Leonel", "apellido" : "Sanchez" }
>
```

Ilustración 15: Ejemplo de búsqueda por coincidencia exacta

El documento *{ _id: 1 }* no es devuelto, ya que no cumple con la condición *{ nombre: 'Leonel' }*. El nombre de los ingenieros se encuentra ubicado en un subdocumento dentro del atributo *ingeniero* y para acceder a el se utiliza la notación de punto *"ingeniero.nombre".*

4.2.4.2 Método findOne()

El método *findOne()* busca y retorna el primer documento que cumpla con *<condición>.*

db.coleccion.findOne({ <condición> })

A modo de comparación, se utilizarán los mismo ejemplos que en el método *find().*

db.ingenieros.findOne({ _id: 1 })

```
> db.ingenieros.findOne( { _id: 1 } )
{
        "_id" : 1,
        "ingeniero" : {
                "apellido" : "Peña",
                "nombre" : "Leonel"
        },
        "lenguajes" : [
                [
                        "Java",
                        "PHP",
                        "C/C++"
                ]
        ]
}
> 
```

Ilustración 16: Ejemplo de búsqueda única a través del identificador de documento

El resultado es el mismo que en *find({_id: 1})*

db.ingenieros.findOne({ nombre: 'Leonel' })

```
> db.ingenieros.findOne( { nombre: 'Leonel' } )
{
        "_id" : ObjectId("51257148633e731e8f251a5a"),
        "nombre" : "Leonel",
        "edad" : 25
}
> 
```

Ilustración 17: Ejemplo de búsqueda única por atributo nombre

Debido a que *findOne()* retorna el primer documento que cumpla con el criterio de búsqueda, sólo devuelve un elemento, el documento encontrado en la Ilustración 17.

El documento seleccionado por *findOne()* es el primero que cumpla la condición, independientemente de su valor *_id*, posición en la inserción (primero, último, etc) o incluso de su ubicación en disco.

La ventaja de este método con respecto a *find()* es que *findOne()* no continua realizando un escaneo sobre los documentos restantes una vez que ya encontró aquel que cumple la condición. Esto resulta eficaz en los casos en que sólo se desea hallar el primer documento que cumpla el criterio sin importar si existe otro.

4.2.5 Eliminar documentos

La eliminación de documentos se realiza a través de la operación *remove*, cuya sintaxis es la siguiente:

db.coleccion.remove(<consulta>)

Por ejemplo, para eliminar todos los documentos que contienen ingenieros de nombre *Leonel*, se utiliza la siguiente instrucción:

db.ingenieros.remove({ 'ingeniero.nombre': 'Leonel' })

```
> db.ingenieros.remove({ 'ingeniero.nombre': 'Leonel' })
> db.ingenieros.find()
{ "_id" : ObjectId("51257148633e731e8f251a5a"), "nombre" : "Leonel", "edad" : 25 }
{ "_id" : 2, "nombre" : "Pedro", "apellido" : "Correa" }
{ "_id" : 3, "nombre" : "Juan", "apellido" : "Castillo" }
{ "_id" : 4, "nombre" : "Leonel", "apellido" : "Sanchez" }
> 
```

Ilustración 18: Ejemplo de eliminación de documentos

La operación anterior eliminó el documento con *{ _id: 1 }*. La Ilustración 12 muestra el mismo listado de documentos que la Ilustración 18 antes de la eliminación.

4.2.6 Indexación

Las colecciones en MongoDB tienen soporte de indexación de documentos y su comportamiento es análogo a los índices de las base de datos relacionales.

La finalidad de los índices es evitar que en cada consulta, *find()* o *findOne()* realicen un escaneo completo de la colección en búsqueda de aquellos documentos que cumplan con el criterio solicitado.

4.2.6.1 Creación de Índices

La sintaxis para la creación de índices es:

db.colección.createIndex(<atributos>)

Por defecto, todas las colecciones tienen un índice sobre el atributo *_id.*

Continuando con el ejemplo del capítulo, la siguiente instrucción crea un índice sobre el atributo *nombre* de la colección *ingenieros.*

db.ingenieros.createIndex({ 'nombre': 1 })

Existen dos valores que puede tomar el atributo indexado: "1" para crear un índice en orden ascendente y "-1" para uno descendente.

También es posible la creación de índices compuestos. La sintaxis es la misma y sólo hay que añadir los demás atributos a indexar.

db.ingenieros.createIndex({ 'nombre': 1, 'edad': -1 })

En este caso el atributo *nombre* está indexado en orden ascendente y *edad* en orden descendente.

4.2.6.2 Evaluando el desempeño

Para evaluar el desempeño de un índice se utiliza el método *explain()* que es agregado al final de una consulta *find*() o *findOne().*

La sintaxis es la siguiente:

db.coleccion.find({<consulta>}).explain()

Esta consulta no mostrará los documentos encontrados por *find*(), sino que explicará en detalle como se utilizaron los índices para encontrar los documentos que cumplen con el criterio de búsqueda.

La Ilustración 19 muestra el resultado de *explain()* antes de crear el índice para el ejemplo del capítulo.

```
> db.ingenieros.find( {nombre: 'Leonel'} ).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 4,
        "nscanned" : 4,
        "nscannedObjectsAllPlans" : 4,
        "nscannedAllPlans" : 4,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {

        },
        "server" : "neox:27017"
} _
```

Ilustración 19: Información de rendimiento de una consulta sin indexación

EL método *explain*() provee de información útil para evaluar el desempeño de las consultas. Los atributos a tomar en cuenta son:

- n: Número de documentos retornados por el método *find().*
- nScannedObjects: Número total de documentos escaneados durante la consulta.
- nScanned: Número de entradas del índice escaneadas durante la consulta.

En la Ilustración 19, n=2 lo cual significa que el método *find*() retornó dos documentos, luego nScannedObjects=4, para encontrar los dos documentos MongoDB tuvo que inspeccionar todos los documentos de la colección (4) y nScanned=4 indica que se revisaron 4 entradas del índice, es decir, el índice completo. Esto no es un buen indicio, ya que si la colección posee cientos de miles de documentos, MongoDB tendrá que inspeccionar uno por uno cada vez que se realice una consulta, tardando mucho tiempo y desperdiciando recursos del sistema.

```
> db.ingenieros.createIndex({ 'nombre': 1 })
> db.ingenieros.find( {nombre: 'Leonel'} ).explain()
{
        "cursor" : "BtreeCursor nombre_1",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 2,
        "nscanned" : 2,
```

Ilustración 20: Información de rendimiento de una consulta indexada

Al indexar el atributo *nombre*, el resultado es mejor y como se aprecia en la ilustración Ilustración 20 el número de objetos escaneados disminuyó. Si la colección tuviera cientos de miles de documentos, el impacto en el rendimiento se apreciaría con más notoriedad.

4.2.6.3 El índice óptimo

El índice óptimo es aquel en que *n = nObject = nObjectScanned.*

El índice creado en el ejemplo anterior, Ilustración 20, es óptimo, pero sólo si lo que se busca es el nombre del ingeniero. ¿Qué sucede si al criterio de búsqueda se le agrega un segundo atributo que no está indexado?

```
> db.ingenieros.find( {nombre: 'Leonel', apellido: 'Sanchez'} ).explain()
{
        "cursor" : "BtreeCursor nombre_1",
        "isMultiKey" : false,
        "n" : 1,
        "nscannedObjects" : 2,
        "nscanned" : 2,
```

Ilustración 21: Información de rendimiento de una consulta parcialmente indexada

Si al criterio de búsqueda se le agrega una segunda condición que no está indexada, el índice es parcialmente útil y MongoDB deberá inspeccionar al interior de los documentos para determinar si el atributo no indexado se encuentra o no y si cumple la condición.[9]

En la Ilustración 21 queda demostrado lo explicado anteriormente y lo que sucede es lo siguiente: MongoDB utiliza el índice sobre el atributo *nombre* para llegar a los dos documentos que lo contienen (nscanned = 2), luego al no existir indexación para *apellido* se debe inspeccionar el contenido de ambos documentos (nscannedObjects = 2) para determinar si cumplen con la condición de igualdad *apellido = 'Sanchez'.* Finalmente, es retornado el único documento que la cumple (n = 1).

En un escenario ideal, todas las consultas que se realicen a la base de datos deben estar apoyadas por índices[10]. Siguiendo esta premisa, la instrucción a continuación crea un índice compuesto para apoyar la consulta por *nombre* y *apellido.*

db.ingenieros.createIndex({ 'nombre': 1, 'apellido': 1 })

```
> db.ingenieros.createIndex({ 'nombre': 1, 'apellido': 1 })
> db.ingenieros.find( {nombre: 'Leonel', apellido: 'Sanchez'} ).explain()
{
        "cursor" : "BtreeCursor nombre_1_apellido_1",
        "isMultiKey" : false,
        "n" : 1,
        "nscannedObjects" : 1,
        "nscanned" : 1,
```

Ilustración 22: Información de rendimiento de una consulta con índice compuesto

Como se aprecia en la Ilustración 22, n = nscannedObjects = nscanned, por lo tanto el índice es un índice óptimo.

Finalmente, la colección ingenieros cuenta con dos índices, los cuales apoyan las consultas por *nombre* y *nombre y apellido.*

4.2.7 Agregación

4.2.7.1 MapReduce

MapReduce es un modelo de programación (framework) paralelo que permite el procesamiento distribuido de grandes conjuntos de datos en un cluster de computadores[1]. Este framework permite la ejecución de complejas consultas de agregación.

La ejecución de una operación MapReduce es realizada por medio de la siguiente instrucción:

db.coleccion.mapReduce(<Funciones>)

Las operaciones en MapReduce están compuestas por un conjunto de funciones escritas en Javascript que son ejecutadas en serie, el orden es el siguiente:

1. Map: Convierte los documentos de una colección en pares clave-valor (parecidos a las tablas hash).
2. Reduce: Reduce la tabla hash devuelta por Map a un sólo valor por clave. Aquí es donde se realizan operaciones de cálculo, tales como: suma, resta, número máximo, estadísticas, etc.
3. Out: Especifica donde ubicar de los resultados de la ejecución de la operación map-reduce.
4. Finalize: Es la encargada de preparar los resultados para ser enviados a la "salida"

Continuando con el ejemplo del capítulo, a continuación se calculará el promedio de edad de los ingenieros.

Antes que nada, se debe agregar la edad de los ingenieros restantes:

db.ingenieros.update({ _id: 2}, { $set: {'edad': 48}})

db.ingenieros.update({ _id: 3}, { $set: {'edad': 32}})

db.ingenieros.update({ _id: 4}, { $set: {'edad': 41}})

```
> db.ingenieros.find()
{ "_id" : ObjectId("51257148633e731e8f251a5a"), "apellido" : "Peña", "edad" : 25, "nombre" : "Leonel" }
{ "_id" : 2, "apellido" : "Correa", "edad" : 48, "nombre" : "Pedro" }
{ "_id" : 3, "apellido" : "Castillo", "edad" : 32, "nombre" : "Juan" }
{ "_id" : 4, "apellido" : "Sanchez", "edad" : 41, "nombre" : "Leonel" }
> 
```

Ilustración 23: Listado de documentos con atributo edad agregado

Lo primero que se debe hacer es crear la función Map, la cual es la encargada de asociar cada atributo *edad* a una clave. Para este caso puntual, la clave debe ser la misma para todos, ya que se calculará el promedio de la edad de todos los documentos.

```
var edadMap = function() {
    emit("promedioEdad", this.edad);
};
var reduceFunction = function(clave, edad) {
    return (Array.sum(edad)/edad.length);
};
db.ingenieros.mapReduce(
    edadMap,
    reduceFunction,
    { out: "ingenieros.estadisticas" }
)
```

```
> var edadMap = function() {
...     emit("promedioEdad", this.edad);
... };
>
> var reduceFunction = function(clave, edad) {
...     return (Array.sum(edad)/edad.length);
... };
>
> db.ingenieros.mapReduce(
...     edadMap,
...     reduceFunction,
...     { out: "ingenieros.estadisticas" }
... )
{
        "result" : "ingenieros.estadisticas",
        "timeMillis" : 7,
        "counts" : {
                "input" : 4,
                "emit" : 4,
                "reduce" : 1,
                "output" : 1
        },
        "ok" : 1,
}
> 
```

Ilustración 24: Ejecución de una operación MapReduce

La función *edadMap* emite hacia la función *reduceFunction* una estructura parecida a una tabla Hash y que contiene los siguientes datos:

{ "promedioEdad"=> 25, " promedioEdad"=>48, "promedioEdad"=>32, " promedioEdad"=> 41}

Luego, la función *reduceFunction* reduce la estructura recibida a un sólo valor por cada clave. Esta es la razón por la cual se utilizó una clave estática para las edades. Por ejemplo, si se hubiese utilizado el identificador de documento como clave, se obtendría una tabla Hash como la siguiente:

{ "51257148633e731e8f251a5a"=> 25, "2"=>48, "3"=>32, "4"=> 41}

Esto es irreducible, ya que existe un sólo valor por clave. Por lo tanto, el resultado de aplicar la función de reducción habría sido la misma tabla hash.

Retomando la tabla hash original, la función *reduceFunction* calcula el promedio y lo envía a la función de salida, que en este caso escribe el resultado en la colección *ingenieros.estadisticas*.

```
> db.ingenieros.estadisticas.find({})
{ "_id" : "promedioEdad", "value" : 36.5 }
>
```

Ilustración 25: Documento que contiene el promedio calculado

Se ha presentado un ejemplo simple el cual cubre los aspectos básicos de MapReduce y como implementar una operación simple de agregación. Las posibilidades con MapReduce son bastas y de las más variadas.

Sin embargo, ésta potente herramienta añade complejidad extra para consultas de agregación simples y comunes, para ello MongoDB posee su propia API de agregación que utiliza MapReduce como base y añade simplicidad en las consultas. Dicha API no es cubierta en este estudio, ya que no presta utilidad al mismo.

4.2.8 Sharding

La escalabilidad horizontal es la funcionalidad básica de muchas bases de datos NoSQL y MongoDB no es la excepción.

El *Sharding* es la característica que permita a MongoDB escalar horizontalmente a través de un cluster, particionando una colección y almacenando diferentes porciones de ella (chunks) en distintas máquinas a través de un *Sharded Cluster.* Esto tiene un efecto de balanceo de carga y de datos, impactando directamente en el rendimiento del cluster. Añadir una nueva máquina al cluster provee mayor capacidad de escritura y almacenamiento.

La operación de distribución de los datos es realizada automáticamente por MongoDB.

4.2.8.1 Compartiendo una colección

Antes de que MongoDB pueda compartir una colección a través del cluster, es necesario activar el sharding de la base de datos que la contiene:

sh.enableSharding("<nombre base de datos>")

Continuando con el ejemplo del capítulo, la instrucción a continuación activa el sharding de la base de datos *prueba* que es donde reside la colección *ingenieros:*

sh.enableSharding("prueba")

Una vez activado el sharding para la base de datos, también se debe hacer para la colección:

```
sh.shardCollection("<base de datos>.<coleccion>", llave-compartida)
```

La llave compartida indica a MongoDB como distribuir los documentos a través del cluster. Esta llave es un atributo presente en cada documento y de acuerdo a su valor, MongoDB distribuirá rangos de ellos a ciertos nodos del cluster.

```
sh.shardCollection("prueba.ingenieros", { "nombre": 1})
```

En este caso, la llave compartida es el *nombre* y todos los nombres que tengan el mismo valor se almacenan en el mismo nodo.

4.2.8.2 La llave óptima

Seleccionar una llave óptima no es sencillo y depende de la estructura de los datos almacenados. La correcta elección de llave definirá el desempeño del cluster y con ella la capacidad de escritura y lectura.

Las llaves compartidas tienen una cardinalidad asociada que está dada por el número de valores que puede tomar un atributo.

Es así como una llave con baja cardinalidad se refiere a la poca variedad de valores que tiene un atributo. Todos los documentos que tienen el mismo valor en la llave, permanecen en el mismo nodo. Esto impacta en el desempeño del cluster, ya que un nodo puede ser sobrecargado y los demás se encuentren desocupados.

En contraparte, una llave con una alta cardinalidad asegura la distribución de los datos, pero no garantiza la escalabilidad de las escrituras ni el aislamiento de las consultas.

Si se utilizan atributos que tienen un incremento constante y monótono, como las fechas o el atributo *_id* generado por MongoDB, todas las inserciones se realizan sobre un sólo nodo, afectando la capacidad de escritura del cluster.

Otro aspecto es el aislamiento de las consultas, que se refiere a la capacidad de MongoDB de encaminar directamente los datos solicitados al nodo que los contiene. Si las consultas no contienen la llave compartida, entonces MongoDB deberá consultar a todos los nodos sobre la ubicación de los datos solicitados, retrasando el tiempo de respuesta.

Una buena llave es aquella que distribuye las escrituras a través de los cluster y permite el correcto direccionamiento de las lecturas.

No existe un método definido para seleccionar la llave óptima, pero si hay fórmulas que pueden ayudar a hacerlo. Una de ellas es *el juego de cartas*[11] cuyo método facilita la elección de una

correcta llave. A continuación se traduce el párrafo en que se explica como elegir una clave según el método anterior.

Utilizar como primera llave el tiempo de los datos, seleccionando aquel que no genere porciones de datos grandes ni muy pequeñas. Por ejemplo, si en un mes se insertan 30GB, entonces el mes es una buena llave { 'mes' : 1 }. Si en un mes se almacena 1 GB, entonces es mejor utilizar el año como llave. En cambio si se obtienen 500GB por mes, un día es una buena elección y si se insertan 5.000GB por segundo, los milisegundos son más adecuados.

{ <tiempo> : 1, <criterio de búsqueda> : 1 }

El segundo parámetro consiste en agrupar los datos por el criterio de búsqueda, como por ejemplo: nombre de usuario, dirección de correo electrónico o cualquier otro dato que maneje la aplicación.

4.2.8.3 Estado del sharding

Utilizando la siguiente instrucción, es posible conocer el balanceo del cluster en cuanto a número de chunks:

```
sh.serverStatus()
```

4.2.9 Caché de MongoDB

El caché de MongoDB es una funcionalidad de la base de datos que hace uso de la memoria del sistema para mantener ciertas porciones de los datos y aumentar la velocidad de acceso a ellos.

Los datos que no son consultados no se mantienen en memoria, y una vez que un dato es consultado es ingresado en ella. Todos los datos deben estar en memoria antes de ser manipulados o accedidos.[12]

5 ANÁLISIS

5.1 Introducción

Este capítulo cubre todos los aspectos necesarios para comprender la composición de los datos a almacenar, las estructuras relacionadas con ellos y su estado actual dentro de MongoDB.

El primer tema que se aborda es el flujo que debe seguir una muestra desde que es tomada hasta que es almacenada, incluyendo su procesamiento dentro del TMC.

Luego, se presenta el formato de las estructuras de datos que contienen las muestras enviadas hasta el TMC como resultado del monitoreo de las antenas.

Finalmente, se expone el diseño actual del esquema utilizado para almacenar los datos de monitoreo, analizando su impacto y consecuencias en el desempeño de las consultas.

5.2 Flujo de Datos

Los Blobbers recolectan las muestras tomadas por componentes de hardware basados en ACS, éstas son encapsuladas como mensajes y enviadas a Apache ActiveMQ, donde son consumidas y procesadas por el software TMC para, finalmente, ser almacenadas en MongoDB.

La Ilustración 26 presenta un diagrama de flujo que describe las actividades realizadas por el TMC para almacenar permanentemente los datos.

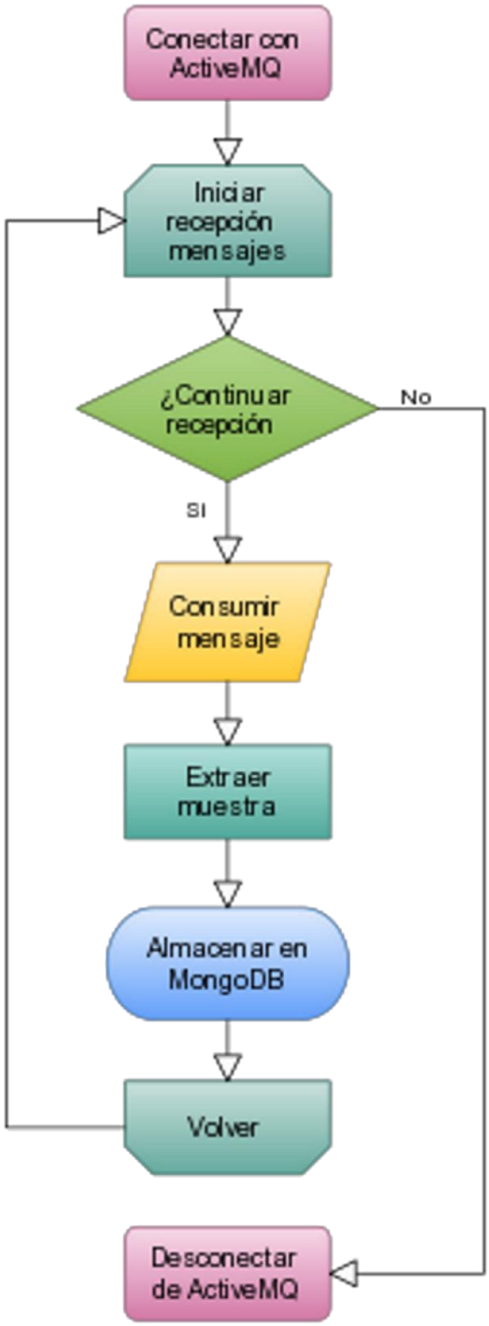


Ilustración 26: Diagrama de flujo del TMC en la obtención, procesamiento y almacenamiento de mensajes

5.3 Información general sobre el monitoreo

ALMA está compuesta, inicialmente, por sesenta y seis antenas que poseen componentes y subcomponentes, los cuales son monitoreados constantemente para asegurar el correcto funcionamiento de todos los dispositivos, y con ello la fidelidad de los datos provenientes de las observaciones espaciales.

Además de las antenas, existen otros elementos del arreglo de ALMA que también son monitoreados y procesados por el software TMC, dentro de ellos se encuentran: las referencias fotónicas, el correlacionador, ArrayTime, WeatherStation, entre otros. Estos elementos son subsistemas ubicados en el edificio central (OSF y AOS) y no forman parte de las antenas, sin embargo, y para todo efecto de almacenamiento de datos de monitoreo, éstos utilizan la misma estructura de datos que ellas.

En total, hay 100.000 puntos de monitoreo y cada uno de ellos tiene su propio tiempo de muestreo, que puede variar de 0,048 a 300 segundos. Esta gran diferencia en el tiempo tiene directa relación con la criticidad para la operación de ALMA que tenga aquel punto de monitoreo. Así, los puntos críticos son monitoreados cada segundo y los menos cada 300 segundos.

De ahora en adelante, la frase *antenas físicas* hará referencia a las antenas propiamente tal y *antena* hará mención a las *antenas físicas* y a los subcomponentes antes mencionados.

5.4 Estructura de un mensaje

Los Blobbers encapsulan los datos monitoreados como mensajes y los introducen en ActiveMQ, Ilustración 2. Estos mensajes son objetos de la clase *MapMessage* del paquete *javax.jms* de la API de Apache ActiveMQ.

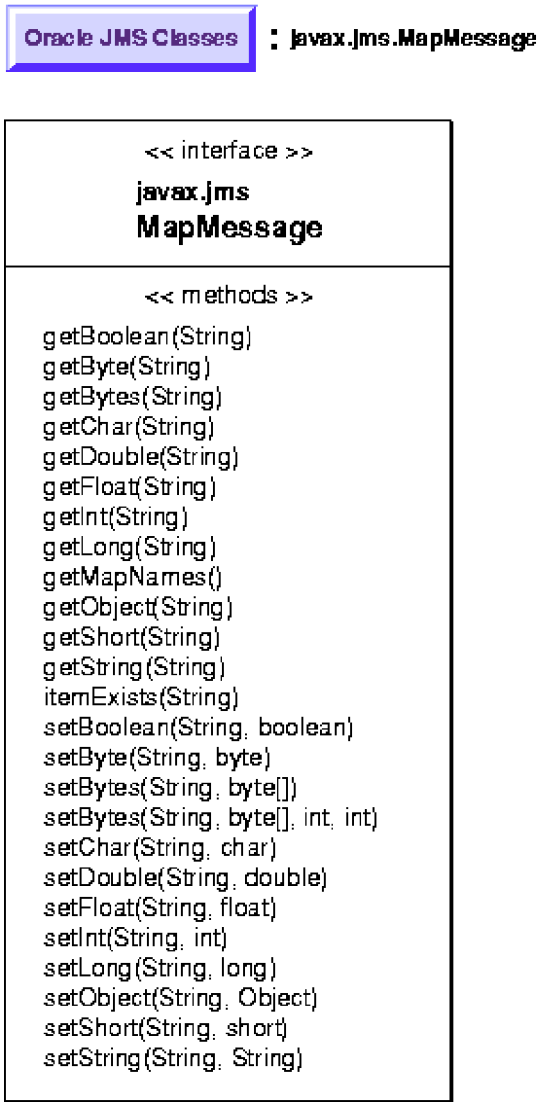


Ilustración 27: Diagrama de clase de MapMessage

La siguiente definición es una traducción de la documentación de MapMessage de Oracle:

> "Un MapMessage es un objeto utilizado para enviar un conjunto de pares nombre-valor. Los nombres son objetos String y los valores son tipos de datos primitivos en el lenguaje de programación Java. Los nombres deben tener un valor distinto de nulo o de una cadena vacía. Las entradas pueden ser accedidas secuencialmente o randómicamente a través del nombre. El orden de las entradas no está definido."[13]

La Tabla 2 explica los pares nombre-valor utilizados por los Blobber para encapsular los datos.

Atributo	Contenido	Ejemplo
componentName	Nombre del componente. El nombre de los componentes está formado por el nombre del subsistema, el nombre de la antena, el nombre del componente (interno de la antena) y, pero no siempre, el nombre del subcomponente.	- CONTROL/CM04/LLC - CONTROL/DV09/FrontEnd/WCA7 - CONTROL/DV10/FrontEnd/Cryostat
propertyName	Nombre de la propiedad. Es utilizada para resolver el nombre del punto monitoreado.	- LO_PHOTOMIXER_VOLTAGE - LO_PHOTOMIXER_CURRENT - POWER_SUPPLY_1_VALUE
clob	Serie de muestras concatenadas que contienen el estado del punto monitoreado.	- 135745302280694810|4.90454 - 135745302379494460|6 - 135745302282272610|-14.713619999999999
serialNumber	Número de serie. Uso interno en ALMA.	- 10bfae3e010800c9
index	Número de índice. Utilizado en conjunto con *propertyName* para resolver el nombre del punto monitoreado.	- 0 - 1 - 2

Tabla 2: Estructura de un mensaje

Los mensajes no incluyen el nombre del punto monitoreado, este debe ser resuelto utilizando el método *getMonitorPointName(String propertyName, int index)* de la clase escrita en java *MonitorPointNameResolver* perteneciente al paquete de utilidades de ALMA *alma.tmcdb.utils.MonitorPointNameResolver*.

5.5 Estructura de una muestra

Una muestra es el escalar del estado de un punto de monitoreo y el tiempo al que corresponde ese estado. Se agrupan en un par tiempo/valor separados por el carácter "|".

```
"135745302282272610|-14.7136199999999999"
```

Ilustración 28: Ejemplo de una muestra

El número antes del carácter "|" corresponde al tiempo de muestreo y el que se encuentra después al valor.

El tiempo se encuentra en formato ACS timestamp el cual es transformado a epoch una vez que la muestra es procesada por el software TMC.

5.6 Estructura y procesamiento de un clob

Los valores propiamente tal del monitoreo de las antenas, son enviados en un formato que agrupa una serie de muestras (pares tiempo/valor) para un solo punto de monitoreo de un componente en un determinado rango de tiempo. Esta serie de muestras se denomina *clob*.

Como se aprecia en la sección *5.4 Estructura de un mensaje*, el clob es enviado dentro de la clase *Message* desde los Blobbers hasta Apache ActiveMQ donde es consumido y procesado por el software TMC.

El procesamiento del clob dentro del software TMC es bastante simple, pero antes de continuar se verá la estructura de un clob.

Como se mencionó anteriormente, un clob está compuesto por una serie de muestras para un solo punto de monitoreo de un componente en un determinado rango de tiempo.

```
"135745302282272610|-14.7136199999999999|135745302382272610|-14.7136199999999999"
```

Ilustración 29: Ejemplo de un clob

Como se aprecia en la Ilustración 29, un clob no es más que la concatenación del tiempo en que se tomó una muestra y el valor del punto monitoreado, ambos separados por el carácter "|". El ejemplo presentado en la Ilustración 29 pertenece al componente "CONTROL/DV06/LO2BBpr2" en el punto de monitoreo "POWER_SUPPLY_3_VALUE" entre "2012-12-11 11:50:28" y "2012-12-11 11:50:38".

Para facilidad del ejemplo, se utilizó un clob de sólo dos muestras, pero no existe límite en el número a utilizar, sin embargo, el número de muestras concatenadas en un clob está dado por el tiempo de

muestreo, que es distinto para cada punto de monitoreo y puede variar desde 0,048 segundo hasta 300 segundos (puede ser modificado en un futuro). Pero eso está fuera del alcance de este estudio, lo único que se debe tener en cuenta es que no hay un límite en el número de muestras por clob.

El procesamiento realizado por el software TMC para obtener las muestras dentro de un *clob* es el siguiente (fragmento de código extraído del software TMC y no fue escrito por el autor de este estudio):

```
1. String clob = message.getString("clob").trim();
2. String[] values = clob.split("\\|");
3. for (int i=0; i < values.length; i = i+2) {
4.         long acsTime_tmp = Long.parseLong(values[i]);
5.         long acsTime = (acsTime_tmp-122192928000000000L)/10000L;
6.         valor = values[i+1];
7. }
```

La explicación se realiza línea por línea:

1. Se extrae el clob del mensaje y se quitan posibles espacios en blanco que pueda traer.
2. El clob es particionado en subcadenas y almacenado en un arreglo.
3. Se itera sobre el arreglo con un incremento de dos unidades, ya que la posición par dentro de el corresponde al tiempo y la impar al escalar del estado muestreado.
4. El tiempo en formato especial es puesto dentro de una variable de tipo entero largo, ya que dentro de la clase *Message* se almacena como cadena de caracteres.
5. Se transforma el tiempo desde el formato especial a epoch.
6. El escalar del punto monitoreado, correspondiente al tiempo transformado anteriormente, está disponible en la posición siguiente dentro del arreglo (i+1). La variable *valor* no es parte del fragmento extraído del software TMC y sólo fue introducida como apoyo visual para explicar la extracción del escalar muestreado.

Finalizado este proceso, el tiempo y el escalar quedan disponibles para ser almacenados dentro de MongoDB.

5.7 Meta-datos

Los meta-datos son todos aquellos campos que proveen información sobre la identificación de las muestras. Es así como la antena, el componente, la propiedad, el punto de monitoreo, tiempo de muestreo, número de serie e índice son meta-datos. Sin embargo, los esenciales son: la antena, el componente y punto de monitoreo, ya que sin ellos no es posible identificar, de ninguna manera, una muestra.


```
{
    "location" : "TFINT",
    "componentName" : "CONTROL/DV06/LO2BBpr2",
    "propertyName" : "POWER_SUPPLY_3_VALUE",
    "monitorPointName" : "POWER_SUPPLY_3_VALUE",
    "serialNumber" : "10bfae3e010800c9",
    "index" : "0"
}
```


Ilustración 30: Meta-datos en formato JSON

Cabe señalar que el tiempo en que fue tomado el estado de un punto de monitoreo no forma parte de los meta-datos, ya que por sí solo no tiene sentido. En cambio, sí lo adquiere cuando se encuentra en conjunto con su escalar, formando una muestra (par tiempo/valor). Esto no quiere decir que deban almacenarse juntos en un mismo campo o atributo.

Complementando el párrafo anterior, los meta-datos siempre se repiten para cada muestra, por ejemplo, el nombre de la antena a la que pertenece una muestra no cambia, aunque ésta sea tomada una y otra vez. Sin embargo, si lo hace el tiempo de la muestra que, efectivamente, aumenta en cada muestreo.

5.8 Esquema de Almacenamiento Actual

Actualmente, los mensajes son almacenados en MongoDB utilizando el esquema de la Ilustración 31

Como se explicó en la sección *4.2.2.1 Documentos* , MongoDB almacena los datos en formato BSON que es una extensión mejorada de JSON.

```
{
    "_id" : { }
    "date" : {}
    "location" : ""
    "componentName" : ""
    "propertyName" : ""
    "monitorPointName" : ""
    "serialNumber" : ""
    "monitorValue" : ""
    "acsTime" : ""
    "index" : ""
}
```

Ilustración 31: Estructura del esquema actual

Todos los datos son almacenados en una sola colección denominada *MonitoringData* que se encuentra dentro de la base de datos *MONDB* en el cluster de pruebas de MongoDB.

Este diseño crea un documento BSON para cada una de las muestras. Por ejemplo, para un punto de monitoreo que es muestreado cada 30 segundos, se genera un nuevo documento BSON cada 30 segundos.

Este enfoque es muy parecido al de las bases de datos relacionales, ya que la colección *MonitoringData* crece como si fuera una tabla a la que se le insertan tuplas[2].

2 Secuencia de valores agrupados como si fueran un único valor que, por su naturaleza, deben ir juntos.

```
{
    "_id" : { "$oid" : "50520ff925d8b6dfb8b4c353" },
    "date" : { "$date" : 1347554984516 },
    "location" : "TFINT",
    "componentName" : "CONTROL/CM12/DRXBBpr1",
    "propertyName" : "POWER_ALARM_REG_B",
    "monitorPointName" : "POWER_ALARM_REG_B_PSUMMARY",
    "serialNumber" : "10bfae3e010800c9",
    "monitorValue" : "255",
    "acsTime" : 135668477845168070,
    "index" : 0
}
```


Ilustración 32: Ejemplo de documento del esquema actual

El número de registros que el software TMC inserte en MongoDB está dado por la cantidad de muestras que contenga el *clob* de un mensaje, de manera que, si un *clob* está compuesto por 3 muestras, entonces serán 3 los objetos BSON que el software TMC insertará en 3 documentos distintos dentro de la colección *MonitoringData* de *MONDB*.

5.8.1 Cifras sobre lo almacenado

Durante los meses de Septiembre, Octubre y Noviembre del año 2012 el software TMC insertó en la colección *MONDB* de MongoDB datos reales provenientes de las antenas.

La Tabla 3 presenta el rango de fechas que está almacenado en cada uno de los meses. Sin embargo, es posible encontrar días en los que no existen datos, esto ocurre por varias razones, pero las principales son:

- Operaciones de mantenimiento de antenas: Constantemente, las antenas son sometidas a operaciones de mantenimiento y muchas veces deben ser desconectadas, provocando que no se envíe información de monitoreo.
- Cortes en el suministro energético: Esto puede ser tanto por fallas en el equipamiento, operaciones de mantenimiento, cambios en los generadores, etc.

- Apagado del cluster: Cambios en los equipos de alimentación energética, reiniciado completo, inundaciones y fallas en el sistema de refrigeración, son sólo algunas de las razones por las cuales se ha apagado el cluster en el pasado.
- Reiniciado de Apache ActiveMQ: Por razones de cambio en las configuraciones, actualización del software o simplemente mantenimiento, se ha reiniciado ActiveMQ o la máquina sobre la cual funciona.

	Septiembre	**Octubre**	**Noviembre**
Rango de datos	Mes completo	Mes completo	Hasta día 25
Número de registros	3.586.119.205	1.572.747.913	585.100.587

Tabla 3: Información mensual de los registros en el esquema actual

Número total de registros	6.048.580.311
Promedio de peso	262 bytes/registro
Total peso datos	1,8 TB (aprox.)
Total peso indices	1,9 TB (aprox.)
Total (datos+indices)	3,7 TB (aprox.)

Tabla 4: Estadísticas de los registros en el esquema actual

Uno de los requerimientos del TMC dice que se generarán 40 GB de datos diarios aproximadamente. Realizando un pequeño cálculo sobre los datos de la Tabla 4, se notará que esto no se cumple. La razón es simple, los 40 GB de datos son para el arreglo completo de antenas, es decir, todas las antenas funcionando y conectadas a la vez.

$$1{,}8\,TB = 1.843{,}2\,GB$$

$$1843{,}2\,GB / 40\text{GB} = 46{,}8\,Antenas$$

Como se aprecia en la estimación, al momento de insertar los datos de la Tabla 4 no se encontraban todas las antenas instaladas.

5.8.2 Uso de meta-datos

En la sección *5.7 Meta-datos* , se explicó que los meta-datos para un punto de monitoreo siempre son los mismos.

Analizando la estructura que presenta el diseño actual, se puede notar que los meta-datos son repetidos en cada inserción de un nuevo documento, generando una gran cantidad de información redundante.

Para ser más exactos, a continuación se presenta una estimación sobre el espacio utilizado por los meta-datos durante los meses de Septiembre, Octubre y Noviembre del año 2012.

Considerando que cada registro tiene un peso promedio de 262 bytes y una muestra (tiempo y escalar) 48 bytes.

$$262\text{bytes} - 48\text{bytes} = 214\,bytes$$

Los meta-datos tienen un peso aproximado de 214 bytes.

Durante los tres meses se almacenaron 6.048.580.311 de registros. Por lo tanto,

$$6.048.580.311 * 214\text{bytes} = 1.294.396.186.554\,bytes$$

expresado en Gigabytes es 1.205,5 GB.

Es un gran volumen de datos y la pregunta que sigue naturalmente es: ¿Será necesario repetir todos esos meta-datos para cada muestra que es almacenada?

5.8.3 Uso de índices

La base de datos *MonitoringData* tiene dos índices (sin considerar el índice por defecto):

1. Campos indexados: { "date": 1, "componentName": 1, "monitorPointName": 1, "propertyName": 1, "location": 1 }
2. Campos indexados: { "componentName": 1, "monitorPointName": 1, "propertyName": 1 }

El tamaño total de los índices es de 1.9TB, que es más de lo que ocupan los 1.8TB de datos, esto ocurre por dos razones:

Primero, la colección MonitoringData tiene dos índices y entre ambos suman un total de 8 atributos indexados, el esquema actual posee 9 atributos (sin considerar _id). El número de atributos indexados es prácticamente el mismo que el esquema al que indexan.

Segundo, el diseño actual está generando un documento por cada muestra y MongoDB crea una entrada (en ambos índices) para cada uno de los documentos, generando índices del mismo tamaño que la colección de datos de monitoreo.

Más allá del tamaño de los índices y considerando lo visto en el capítulo , a continuación se realiza una prueba para determinar el desempeño de los índices.

```
db.monitorPoints.findOne( {
        "date": { '$gte': ISODate("20121004"), '$lte': ISODate("20121020") },
        "componentName" : "CONTROL/DV10/LLC",
        "monitorPointName": "P_DET",
        "propertyName" : "P_DET"
},
 { _id:1 }).explain()
```

```
"cursor" : "BtreeCursor comp_mpname_proper_IDX",
"n" : 107849,
"nChunkSkips" : 0,
"nYields" : 10710,
"nscanned" : 735106,
"nscannedAllPlans" : 1084863,
"nscannedObjects" : 735106,
```

Ilustración 33: Desempeño del índice en el esquema actual

Como se aprecia en la Ilustración 33, n=107.849, nscanned=735.106 y nscannedObjects= 735.106. El índice óptimo, *4.2.6.3 El índice óptimo*, es aquel en que sólo se inspeccionan los mismos elementos retornados. Esta es la razón por la cual las consultas toman tanto tiempo, ya que MongoDB debe inspeccionar un volumen que es 7 veces mayor al número de documentos devueltos. Forzando a un comportamiento que está lejos de ser el óptimo.

5.8.4 Estado del Sharding

Un aspecto importante en un cluster de MongoDB, es conocer el estado del balanceo de datos.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
  shards:
        {  "_id" : "mongodb1",  "host" : "mongodb1.osf.alma.cl:27018",  "maxSize" : NumberLong(3000000) }
        {  "_id" : "mongodb2",  "host" : "mongodb2.osf.alma.cl:27018",  "maxSize" : NumberLong(3000000) }
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "MONDB",  "partitioned" : true,  "primary" : "mongodb1" }
                MONDB.monitorPoints chunks:
                                mongodb1        9561
                                mongodb2        9562
                        too many chunks to print, use verbose if you want to force print
```

Ilustración 34: Estado del sharding en el diseño actual

Como se aprecia en la Ilustración 34, *mongodb1* y *mongodb2,* prácticamente, tienen el mismo número de porciones de datos (chunks), 9.561 y 9.562 respectivamente. Esto quiere decir que poseen la misma cantidad de rangos de datos asignados. Sin embargo, esto no quiere decir que mongodb1 tiene menos documentos que mongodb2, de hecho, es todo lo contrario.

```
mongos> db.stats()
{
        "raw" : {
                "mongodb1.osf.alma.cl:27018" : {
                        "db" : "MONDB",
                        "collections" : 3,
                        "objects" : 3459264661,
                },
                "mongodb2.osf.alma.cl:27018" : {
                        "db" : "MONDB",
                        "collections" : 4,
                        "objects" : 2589318620,
```

Ilustración 35: Número de documentos por nodo en el diseño actual

Como se aprecia en la Ilustración 35, *mongodb1* tiene casi cien millones de documentos más que *mongodb2.* Esto ocurre porque ciertos puntos de monitoreo tienen mayor actividad (tiempo de muestreo) insertando un mayor número de documentos en el nodo que le corresponde.

5.9 Conclusión

Luego de analizar minuciosamente el esquema actual de almacenamiento se ha concluido que es necesaria una mejora sustancial en el diseño para poder cumplir con el requerimiento número 4 del TMC, el cual dice: *"Tiempo de extracción requerido para 1 variable con datos de 24 horas tiene que ser menor a 1 segundo".* Dicha solución puede ser planteada de una manera radicalmente diferente o simplemente modificando la forma de organización de los datos.

Un aspecto clave que debe ser considerado en dicha solución es la indexación. Los índices son para acelerar las consultas y no para retrasarlas, claramente esta premisa no se está cumpliendo y se debe tener en cuenta para apoyar con el cumplimiento del requerimiento número 4 del TMC.

6 DISEÑO

6.1 Introducción

Ya analizada y comprendida la situación actual del almacenamiento permanente del TMC, se está en condiciones de iniciar el diseño de nuevos esquemas que logren cumplir con los requerimientos.

Se han abordado tres perspectivas de almacenamiento, la primera de ellas consiste en ordenar los datos en múltiples colecciones con la finalidad de atenuar la dispersión e reducir los índices.

El segundo diseño plantea una disminución en el número de documentos generados, por medio de un esquema que se ajusta a la naturaleza de los mensajes.

Finalmente, se radicaliza la postura frente a la cantidad de documentos, para lo cual el tercer diseño adopta un esquema más complejo y sin disección de datos.

6.2 Diseño físico de la base de datos

6.2.1 Múltiples colecciones

La primera idea sobre como disminuir los tiempos de consulta nace de la mano de la organización de los datos, básicamente, reorganizar la ubicación de cada uno de ellos para acotar el universo de búsqueda en cada consulta. Este enfoque no modifica el esquema actual de la base de datos.

En primera instancia, esta reorganización de los datos se puede pensar de las siguientes formas:

1. Una colección diaria: Organizar los 100.000 puntos monitoreo en una colección diaria. Se crearía una colección diariamente y 30 mensualmente.
2. Una colección diaria por antena: Organizar los 100.000 puntos monitoreo en sus respectivas antenas utilizando una colección diaria para cada uno de ellas. Se crearían 70 colecciones diariamente y 2.100 mensualmente.
3. Una colección mensual por antena: Organizar los 100.000 puntos monitoreo en sus respectivas antenas utilizando una colección mensual para cada uno de ellas. Se crearían 70 colecciones mensualmente.
4. Una colección mensual por punto de monitoreo: Organizar los 100.000 puntos monitoreo en una colección para cada uno de ellos. Se crearían 100.000 colecciones mensualmente.

Como se explicó en la sección *4.2.2.2 Colecciones*, sin la utilización de un GridFS no es posible crear más de 24.000 colecciones por base de datos. Por lo tanto, la opción 4 queda descartada.

```
{
    "_id" : { }
    "date" : {}
    "location" : ""
    "componentName" : ""
    "propertyName" : ""
    "monitorPointName" : ""
    "serialNumber" : ""
    "monitorValue" : ""
    "acsTime" : ""
    "index" : ""
}
```

Ilustración 36: Esquema "Múltiples colecciones"

Debido a que este diseño de reorganización de los datos tiene el mismo esquema que el actual, las estadísticas relacionadas con número de documentos por día, número de documentos por punto de monitoreo, tamaño promedio de los documentos, entre otros, se mantienen idénticos. Las únicas variaciones son las expuestas por la ideas número 1, 2 y 3, las cuales son analizadas a continuación:

Ventajas:

- Cada colección tendrá su propio índice. Indices más pequeños es equivalente a menos datos donde buscar lo que incurre en un menor tiempo de búsqueda.
- Granularidad de los respaldos, un colección por antena es más conveniente de exportar/importar, almacenar y respaldar.

Desventajas:

- El número de documentos se mantiene igual.
- El tamaño de los meta-datos se mantiene igual.

Cabe recordar, y como se ha explicado en la sección *4.2.2.3 Esquema flexible*, la creación de las bases de datos y colecciones es realizada automáticamente por MongoDB en tiempo de inserción. Por lo tanto, el software TMC sólo debe enviar los datos para que éstas sean creadas.

6.2.2 Un documento por mensaje

Tanto en el diseño actual como en el de *Múltiples colecciones*, las muestras de los *clob* son separadas en unidades y almacenadas en la base de datos. Este enfoque, que por lo demás es muy parecido (por no decir idéntico) al de una base de datos relacional, no logra explotar la cualidad de los *clob* de transmitirse en series de tiempo, en otras palabras, un *clob* está constituido por la concatenación de varias muestras. Entonces, ¿para qué dividirlas?

Si se pensara en un diseño que se ajuste tal cual a la estructura de los mensajes, es decir, sin separar las muestras de los *clob*, se obtendría una reducción en el número de documentos insertados y con ello un posible aumento en el rendimiento de las consultas.

```
{
    "_id" : ""
    "dateStart" : ""
    "dateEnd" : ""
    "location" : ""
    "componentName" : ""
    "propertyName" : ""
    "monitorPointName" : ""
    "serialNumber" : ""
    "clob" : ""
    "index" : ""
}
```

Ilustración 37: Esquema "Un documento por mensaje"

Considerando un promedio de 10 muestras por clob, a continuación se presenta una estimación, basada en los números expuestos en la sección *5.8.1 Cifras sobre el almacenado* para el mes de Septiembre, sobre los volúmenes de datos generados por el diseño expuesto:

Número total de registros mensuales:

$$3.586.119.205/10 = 358.611.920$$

La disminución en el número de documentos es de un 10% con respecto al esquema actual.

Este diseño no sólo disminuye la cantidad de documentos, sino que también el uso del almacenamiento físico, la razón es simple: menos cantidad de meta-datos. Al utilizar un documento por mensaje se disminuye la redundancia de ellos, que en ambos diseños anteriores eran repetidos para cada muestra. En este diseño, la redundancia es por mensaje, lo que otorga, en promedio, una disminución de 9 meta-datos por documento.

A continuación se presentan las ventajas y desventajas asociadas al diseño:

Ventajas:

1. Menos meta-datos: Los *clob* ya no son particionados, ahorrando espacio físico por medio de la disminución de la redundancia del nombre del componente, nombre de propiedad, ubicación, número de serie e índice.
2. Menos índices: Desde el momento en que hay menos documentos que indexar, los índices se vuelven más pequeños, disminuyendo el tiempo de consulta.
3. Menos búsquedas: Al momento de buscar un mensaje sólo se necesita consultar el rango de fechas que contiene su clob. Sin tener la necesidad de iterar por cada muestra de dato. Esto reduce la cantidad de comparaciones en x veces, dónde x es la cantidad de muestras agrupadas por clob.

Desventajas:

1. Procesamiento del clob: La disección y posterior procesamiento del clob deberá ser realizado en lado del cliente.
2. Meta-datos extras: Debido a que el clob se almacena sin procesar, no es posible conocer los tiempos exactos de todas las muestras contenidas en el. Es por ello, que es necesario añadir al esquema dos atributos más para poder identificar las fechas que acotan al clob, sin extraerlo ni procesarlo. Estos atributos son *dateStart* y *dateEnd*.

```
{
    "_id" : { "$oid" : "50520ff925d8b6dfb8b4c353" }
    "dateStart" : "2012-12-11 11:50:28"
    "dateEnd" : "2012-12-11 11:50:38"
    "location" : "TFINT"
    "componentName" : "CONTROL/DV06/LO2BBpr2"
    "propertyName" : "POWER_SUPPLY_3_VALUE"
    "monitorPointName" : "POWER_SUPPLY_3_VALUE"
    "serialNumber" : "10bfae3e010800c9"
    "clob" : "135745302282272610|-14.713619999999999|135745302382272610|-14.713619999999999"
    "index" : "0"
}
```


Ilustración 38: Ejemplo de un documento al interior del diseño "Un documento por mensaje"

La consulta de búsqueda debe implementar el filtrado de fechas, la versión o tiempo de las muestras, que contiene el clob del mensaje, está acotado por *dateStart,* para la fecha de inicio, y *dateEnd,* para la fecha de término. De esta manera, se puede verificar si las muestras buscadas son un subconjunto parcial o completo de las muestras contenidas en el mensaje.

Con la finalidad de aprovechar las ventajas expuestas en el diseño de *Múltiples colecciones.* Este diseño de *Un documento por mensaje* tiene tres posibilidades para la organización de los mensajes:

1. Almacenar los documentos en una sola colección mensual, diseño actual.
2. Almacenar los documentos en una colección por antena, funcionalidad propuesta en el diseño *Múltiples colecciones*.
3. Almacenar los documentos en una colección diaria, funcionalidad propuesta en el diseño *Múltiples colecciones*.

6.2.3 Un documento diario por punto de monitoreo

Los diseños anteriormente presentados (incluido el actual), insertan una gran cantidad de documentos para almacenar sólo una pequeña variación en cada punto de monitoreo: la muestra. Esto no sería un problema si las consultas se ejecutaran en el tiempo requerido, pero no es el caso y toman mucho más del tiempo aceptable.

El diseño *Múltiples colecciones* entregó una visión más organizacional que una solución real a la problemática, en cambio el diseño *Un documento por mensaje* consideró una reforma en la estructura de almacenamiento de los datos, modificando el esquema y provocando una disminución en la cantidad de documentos.

Sin embargo, ninguno de ellos logra obtener el potencial sobre las cualidades con las que fue creado MongoDB.

Una de estas cualidades, es derivada del formato JSON (base para el formato BSON) y que consiste en la anidación de objetos. Es decir, almacenar un objeto JSON dentro de un atributo de otro objeto JSON. Característica tratada en la sección *4.2.2.1 Documentos*.

Otra de esas cualidades, es la capacidad de agregar atributos a un documento BSON en tiempo de actualización. Característica tratada en detalle en la sección *4.2.3.2 Operación update*.

Para obtener provecho de la capacidad de los objetos BSON de anidar otros objetos en sus atributos, se ha diseñado un esquema, utilizando como base el ejemplo propuesto en la documentación oficial de MongoDB[14], que almacena todos las muestras en un documento diario por punto de monitoreo. Esto significa crear 100.000 documentos diariamente.

El enfoque de este diseño, Ilustración 41, consiste en anidar las muestras utilizando como llave de acceso las horas, minutos y segundos. Dentro del atributo *hourly* se crea un objeto BSON que consta de una serie de números del 0 al 23, los cuales representan las horas del día. Luego, a cada hora del día se le agrega un nuevo objeto BSON con una serie de números del 0 al 59, los cuales representan los minutos. Finalmente, a cada uno de los minutos se le agrega un nuevo objeto BSON con una serie de números del 0 al 59, los cuales representan los segundos.

Los campos que representan los segundos son los que tienen asignado el escalar muestreado de un punto de monitoreo, es así como las horas, minutos y segundos son utilizados como índices para referirse a los escalares almacenados en el subdocumento más interno del atributo *hourly*.

Las muestras son insertadas por el TMC a medida que llegan desde ActiveMQ, utilizando la operación upsert, *4.2.3.3 Operación upsert*.

Por ejemplo, para obtener el escalar de una muestra que fue tomada a las 21:36:54 horas se utiliza la notación *hourly.21.36.54*

Esta forma de anidación de objetos BSON utilizando el tiempo no es una selección azarosa y tiene un sustento computacional que es explicado en la documentación de MongoDB. A continuación se presenta la traducción de dicho sustento.

MongoDB almacena los documentos BSON como una secuencia de campos y valores, no como una tabla hash. Por lo tanto, escribir en el campo número 0 es considerablemente más rápido que en el 1439, ya que MongoDB debe saltar las 1439 entradas antes que el.

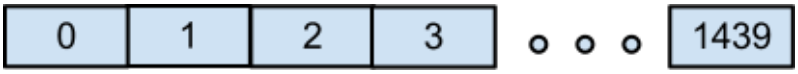


Ilustración 39: Estructura de almacenamiento para un documento que representa los minutos del día

Es por ello, que para optimizar las operaciones de inserción y actualización se ha utilizado una jerarquía de documentos anidados, la cual hace al desempeño de las actualizaciones más uniforme y rápido al final del día.

Ilustración 40: Estructura de almacenamiento para un documento que ha anidado los minutos del día en horas

De esta manera, si se desea acceder al campo 1439, MongoDB salta a través de las 23 horas y luego salta 59 minutos. Dando como resultado, un total de 82 campos saltados versus los 1439 campos saltados en la Ilustración 39.

```
{
    _id : "20120901/CM12/DRXBBpr1/POWER_ALARM_REG_B_PSUMMARY",
    metadata : {
        date : "2012-09-01",
        antenna : "CM12",
        component : "DRXBBpr1",
        property : "POWER_ALARM_REG_B",
        monitorPoint : "POWER_ALARM_REG_B_PSUMMARY",
        location : "TFINT",
        serialNumber : "10bfae3e010800c9",
        index : 0,
        sampleTime : 1 },
    hourly : {
        "0" : {
            "0" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            "1" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            ...
            "59" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 } },
        "1" : {
            "0" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            "1" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 },
            ...
            "59" : {
                "0" : 12345,
                "1" : 12345,
                "2" : 12345,
                ...
                "59" : 12345 } } }

}
```

Ilustración 41: Esquema "Un documento diario por punto de monitoreo"

A continuación se realiza una estimación de los datos generados por el diseño:

Siguiendo la misma línea de las estimaciones realizadas en los anteriormente, los cálculos son en base a 100.000 puntos de monitoreo y 30 días mensuales.

	Número de Documentos
Diariamente	100.000
Mensualmente	3.000.000

Tabla 5: Estimación del número de documentos en "Un documento diario por punto de monitoreo"

Como se aprecia en la estimación, sólo se obtendrán 100.000 documentos diarios y 3.000.000 mensuales. La diferencia con los 3.586.119.205 documentos mensuales del diseño actual es drástica y sin considerar que esa cifra no contiene a todas las antenas, lo cual aumentaría aún más la distancia.

A continuación se presentan las ventajas y desventajas de este enfoque:

Ventajas:

1. Reducción en el tiempo de búsqueda: Al solicitar muestras para un día en específico y sea cual sea el rango de tiempo requerido, MongoDB tiene que obtener solamente un documento y no buscar cientos o miles de ellos. ¿Y si la consulta requiere de dos días?, pues sólo serán dos documentos.
2. Menos entradas en los índices: Se indexarán 100.000 documentos diariamente, dando como resultando índices más pequeños, lo que impactará positivamente en los tiempos de consulta.
3. Disminución del uso de almacenamiento físico: Ahorro sustancial en el uso de espacio en disco, al redundar sólo diariamente y por punto de monitoreo los meta-datos, y no en cada muestra como en el diseño actual.

Desventajas:

1. Relocalización de documentos: Debido a que las muestras son insertadas en tiempo de actualización, el documento en el que se encuentran comienza a crecer y a medida que crece, requerirá más espacio del que es provisto por el bloque físico en el que está almacenado, obligando a MongoDB a re-ubicarlo. Esta situación es tratada en detalle en la sección *6.2.3.1 Relocalización de documentos*.

Con la finalidad de aprovechar las ventajas expuestas en el diseño de *Múltiples colecciones. Un documento diario por punto de monitoreo* tiene dos posibilidades para la organización de los mensajes:

1. Almacenar los documentos en una sola colección diaria: *Múltiples colecciones*.
2. Almacenar los documentos en una sola colección mensual: diseño actual.

6.2.3.1 Relocalización de documentos

Debido a que las muestras son insertadas en tiempo de actualización,*4.2.3.3 Operación upsert*, el documento en el que se encuentran comienza a crecer y a medida que crece requerirá más espacio del que es provisto por el bloque físico en el que está almacenado, obligando a MongoDB a re-ubicar el documento en un nuevo bloque que cuente con el espacio necesario.

Las operaciones de relocalización tienen un alto costo computacional que es pagado por el rendimiento en escritura, ya que MongoDB no insertará el valor enviado hasta que su documento se encuentre reubicado.

Considerando que la relocalización de documentos merma la capacidad de insertar muestras a la base de datos, se está frente a un problema que debe ser resuelto.

6.2.3.2 Pre-ubicación de documentos

La documentación de MongoDB es clara al proponer una solución que consiste en pre-ubicar los documentos con todos los atributos necesarios antes que se inicie la actividad de escritura en ellos.

El ejemplo propuesto por la documentación, consiste en almacenar el número de visitas por página. Los atributos que ellos utilizan son estáticos, si nadie visita la página las estadísticas del tiempo quedan en 0.

Para el TMC, este ejemplo es parcialmente útil. Y se agrega una nueva complejidad, los puntos de monitoreo no tienen tiempo de muestreo estático. Dificultando la creación de los atributos, ya que ¿Cómo saber en qué momento se tomará la muestra de un punto de monitoreo?

Si bien es cierto, es posible conocer de antemano el tiempo de muestreo, no así su serie. Ésto sólo es factible una vez que los datos han sido consumidos y procesados por el TMC, en ese momento se está en condiciones de estimar la serie de tiempo de un punto de monitoreo utilizando la diferencia de tiempo entre la primera muestra del día y la segunda, permitiendo la creación de un documento con

los atributos correspondientes a la serie.

Para evitar que el TMC inserte en una sola operación la totalidad de los documentos diarios, ellos son insertados a medida que llegan, repartiéndolos durante las horas del día. Aún así, durante las primeras horas del día, MongoDB recibirá una carga adicional, pero que se compensa con el rendimiento de escritura durante el resto del día.

Otro aspecto que añade complejidad, es el tratamiento de los atributos no utilizados. En el caso de que los documentos se pre-ubicaran, puede que muchos atributos no sean actualizados ocupando espacio en disco. Esto también es abordable, una vez que los datos sean ingresados se recorre el documento completo eliminando los atributos no utilizados. Pero la situación es la misma, si antes se debía re-ubicar un documento para buscar más espacio, ahora se tendrá espacio intermedio sin utilizar generando fragmentación en el disco y aumentando la carga de MongoDB debido a la constante operación de limpieza de atributos.

Las posibilidades son las siguientes:

1. Pre-ubicar los documentos con un tiempo de muestreo variable, calculado con la diferencia del tiempo de las primeras muestras: Pre-ubicar los documentos con un tiempo variable permite a los atributos del documento ajustarse a las muestras que llegan. Uno de los inconvenientes de este método es que si un punto de monitoreo es deshabilitado y luego habilitado es posible que la serie de su tiempo de muestreo no coincida con los tiempos cálculos al principio del día, haciendo que MongoDB deba re-ubicar los documentos en una eventual falta de espacio físico del bloque.
2. Pre-ubicar los documentos con un tiempo de muestreo fijo: Un documento pre-ubicado con todos los segundos del día (86.400 en total) y un valor de monitoreo estándar de 5 dígitos ocupa un espacio de 1,3 MB en MongoDB. Considerando que diariamente se crean 100.000 documentos, se obtiene un uso de 126,9 GB/día. Mensualmente sería un total de 3.808,6 GB. Este valor es mucho mayor al utilizado actualmente.
3. No pre-ubicar documentos y probar el rendimiento de la escritura: No pre-ubicar los documentos e insertar las muestras a medida que llegan es el método en que se evitan ambos casos anteriores. Sin embargo se obtiene el inconveniente descrito en el capítulo .

Si se considera que el espacio en disco no es un factor decisivo en el diseño a utilizar, y siempre y cuando no afecte al rendimiento, entonces el mejor enfoque son los documentos con tiempo de muestreo fijo.

6.2.3.3 Optimización del TMC

En la capa de aplicación correspondiente al TMCMongoArchiver, es posible realizar una optimización en la forma de insertar las muestras dentro del diseño *Un documento por diario por punto de monitoreo*.

Al momento de particionar el clob dentro del TMC, en vez de insertar individualmente cada una de las muestras extraídas del mensaje, es posible de realizar en una sola operación, es decir, todas las muestras juntas. De esta manera se inserta, en una sola instrucción, 4, 5, 5, 6 o 7 muestras en un mismo documento, aumentando el rendimiento.

Durante el desarrollo de este trabajo de tesis, se escribió un código para importar los datos de monitoreo de los meses de Septiembre, Octubre y Noviembre del 2012, ver sección *7.2 Importación de los datos*. Dentro de esta aplicación, se diseñó e implementó una clase denominada *MongoManager,* que es la encargada de gestionar la conexión con MongoDB, y que provee de dos métodos para realizar las operaciones *upsert.* El primero, inserta una muestra a la vez y el segundo, inserta un bulto de muestras.

6.3 Seleccionando el diseño

En esta sección, se realiza una comparativa de los diseños presentados en las secciones anteriores, con la finalidad de clarificar las diferencias entre ellos y dar las razones cuantitativas de porqué se somete a prueba uno y no otro.

La Tabla 6 presenta una comparación entre los cuatro diseños, utilizando como variable comparativa el número de documentos generados mensualmente.

	Mensualmente
Diseño actual	3.586.119.205
Múltiples colecciones	3.586.119.205
Un documento por mensaje	358.611.920
Un documento diario por punto de monitoreo	3.000.000

Tabla 6: Comparación de los diseños por cantidad de documentos

La Tabla 7 presenta una comparación entre los distintos diseños, utilizando como variable comparativa el peso promedio de los documentos.

	Promedio de Peso (Bytes)
Diseño actual	262
Múltiples colecciones	262
Un documento por mensaje	364
Un documento diario por punto de monitoreo (documento con todos los segundos del día)	1.363.148,8

Tabla 7: Comparación de los diseños por peso promedio de los documentos

Como se aprecia en la Tabla 7, el volumen por documento en el diseño *"Un documento diario por punto de monitoreo"* es superior a los demás diseños, dejando la incógnita ¿Afecta el tamaño de los documentos en el rendimiento?.

Por otra parte, el uso de un documento diario por punto de monitoreo reduce en un 99,91% el número de documentos mensuales, Tabla 6. Es por lo anterior, que el diseño que será sometido a pruebas es *"Un documento diario por punto de monitoreo".*

6.4 Diseño de consultas

Ya seleccionado el diseño, ahora corresponde la creación de las operaciones de inserción/actualización, selección, indexación y sharding.

6.4.1 Inserción / Actualización

La sentencias de inserción / actualización son:

Una muestra:

```
db.monitorData_[mes].update(
        { _id: '20130203/DV10/LLC/P_DET' },
        { $set: { "hourly.1.1.1" : "98765" }},
        { upsert : true }
};
```

Múltiples muestras:

```
db.monitorData_[mes].update(
        { _id: '20130203/DV10/LLC/P_DET' },
        { $set: { "hourly.1.1.1" : "98765", "hourly.1.1.2" : "98765", "hourly.1.1.3" : "98765" }},
        { upsert : true }
};
```

6.4.2 Selección

La siguiente consulta retorna el escalar correspondiente al segundo "18" del minuto "29" de la hora "15".

```
db.monitorData_[mes].findOne( {
        "metadata.antenna": "DV10", "metadata.monitorPoint": "GATE_VALVE_STATE",
        "metadata.component": "FrontEnd/Cryostat", "metadata.date": "2012-9-15" },
        { 'hourly.15.29.18': 1 }
);
```

La siguiente consulta retorna todos los escalares contenidos en el minuto "29" de la hora "15".

```
db.monitorData_[mes].findOne( {
        "metadata.antenna": "DV10", "metadata.monitorPoint": "GATE_VALVE_STATE",
        "metadata.component": "FrontEnd/Cryostat", "metadata.date": "2012-9-15" },
        { 'hourly.15.29': 1 }
);
```

La siguiente consulta retorna todos los valores contenidos en la hora "15".

```
db.monitorData_[mes].findOne( {
        "metadata.antenna": "DV10", "metadata.monitorPoint": "GATE_VALVE_STATE",
        "metadata.component": "FrontEnd/Cryostat", "metadata.date": "2012-9-15" },
        { 'hourly.15': 1 }
);
```

La siguiente consulta retorna un documento completo:

```
db.monitorData_[mes].findOne( {
        "metadata.antenna": "DV10", "metadata.monitorPoint": "GATE_VALVE_STATE",
        "metadata.component": "FrontEnd/Cryostat", "metadata.date": "2012-9-15"
});
```

6.4.3 Índices

```
db.monitorData_[mes].ensureIndex( {
        "metadata.antenna" : 1, "metadata.monitorPoint" : 1,
        "metadata.component" : 1, "metadata.date" : 1
} );
```

6.4.4 Sharding

La llave compartida está compuesta por dos atributos: *antenna y monitorPoint*.

```
sh.shardCollection(
        "OneMonitorPointPerDayPerDocument.monitorData_[mes]",
        {"metadata.antenna":1, "metadata.monitorPoint":1}
);
```

6.5 Postprocesamiento de los datos

Con el objetivo de apoyar la obtención de información estadística acerca de las muestras recolectadas durante un día, se ha diseñado una función de agregación utilizando el framework MapReduce, en la cual se calcula: valor mínimo, valor máximo, promedio y desviación estándar del escalar del punto monitoreado.

El código está adjunto en el apéndice VIII – Postprocesamiento de datos.

La función crea una nueva colección llamada *monitorData_[mes].stats* con una correlación de uno a uno con los documentos, es decir, que por cada documento con datos se genera otro documento en la colección *stats* con los valores estadísticos. Los documentos generados contienen el mismo *_id* que el documento que contiene los datos.

6.6 Conclusión

Se han expuesto varios diseños de posibles esquemas, que cubren desde lo más básico, desde una reorganización de los datos hasta un enfoque más complejo, como es la anidación de varios subdocumentos.

También se han realizado las estimaciones pertinentes para determinar, cuantitativamente, qué diseño es más probable que cumpla con los requerimientos.

Claramente, el diseño *"Un Documento diario por punto de monitoreo"* es el que obtiene las mayores ventajas por sobre los demás, destacando su gran disminución en el número de documentos generados, y con ello, respondiendo a la interrogante planteada al final de la sección *5.8.2 Uso de meta-datos*, ¿Será necesario repetir todos esos meta-datos para cada muestra que es almacenada?

Y la respuesta es no, no era necesario redundar los meta-datos a nivel de muestras.

7 IMPLEMENTACIÓN DE LA SOLUCIÓN

7.1 Instalación de MongoDB

Esta sección cubre los aspectos necesarios para la instalación de MongoDB en una sola máquina, ya muchas de las pruebas fueron implementadas localmente antes de ser trasladadas al cluster de ALMA.

Sin embargo, la puesta en marcha de un sharded cluster no será cubierta por este estudio, a pesar de que el cluster de ALMA es un sharded cluster, dicha instalación cubre muchos aspectos que no son necesarios para la implementación de este trabajo y lo único que lograrían es desviar del objetivo de este estudio. Todo lo necesario para obtener provecho de las cualidades de un sharded cluster son tratadas en la sección *4.2.8 Sharding*.

7.1.1 Sistema Operativo

La instalación de MongoDB fue realizada sobre el sistema operativo Debian Squeeze, distribución de GNU/Linux, con arquitectura de 64 bytes.

La instalación sobre otras arquitecturas y sistemas operativos no será cubierta en este estudio, ya que la documentación de MongoDB es completa y cubre todos esos aspectos.

7.1.2 Configuración del sistema de administración de paquetes (APT)

El sistema de administración de paquetes de Debian (APT) es una herramienta que gestiona (instalar, actualizar, configurar, eliminar) los paquetes *.deb*.

Para asegurar la autenticidad de dichos paquetes *.deb,* APT utiliza un sistema de firmas digitales que se encarga de verificar que las aplicaciones instalas vía APT pertenecen a quienes dicen ser, evitando la infiltración de código malicioso.

Primero, agregar la clave pública GPG del repositorio de 10gen al sistema operativo, para ello abrir una terminal, ingresar como root y escribir:

apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10

Luego, crear el fichero *10gen.list* en el directorio */etc/apt/sources.list.d/* con la siguiente linea:

deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen

Finalmente, actualizar la lista de paquetes de APT a través del siguiente comando:

apt-get update

7.1.3 Instalación de paquetes

Para instalar la última versión estable de MongoDB utilizar el siguiente comando:

apt-get install mongodb-10gen

Una vez que el comando termine, MongoDB se encuentra instalado satisfactoriamente.

7.1.4 Configurando MongoDB

Al igual que la gran mayoría de los paquetes de las distribuciones GNU/Linux, MongoDB utiliza el método de configuración por ficheros de texto ubicados en el directorio */etc*.

Configurar MongoDB, es tan simple como abrir el fichero */etc/mongodb.conf,* modificar los valores deseados y reiniciar la base de datos utilizando el comando:

/etc/init.d/mongodb restart

De esta manera, MongoDB adoptará el comportamiento deseado.

Este estudio no requiere de ninguna configuración especial y sólo basta con el comportamiento por defecto.

7.1.5 Controlando MongoDB

Debido a que MongoDB posee procesos en segundo plano, no implementa una aplicación de escritorio en la cual sea posible controlar su ejecución. Este proceso es realizado a través de comandos con los cuales es posible iniciar, detener y reiniciar la ejecución de la base de datos.

Iniciar:

/etc/init.d/mongodb start

Detener:

/etc/init.d/mongodb stop

Reiniciar:

/etc/init.d/mongodb restart

7.1.6 Utilizando MongoDB

MongoDB provee una interfaz de línea de comandos para su uso, en la cual se pueden realizar inserciones, actualizaciones, eliminaciones y búsquedas de documentos, además de tareas administrativas del cluster o nodo en cuestión.

El programa *mongo* es el encargado de abrir la interfaz de comunicación, para ello simplemente se debe escribir el comando *mongo* en la linea de comandos de Debian. Una vez dentro, se puede ver el carácter ">" al lado izquierdo del cursor, ésto indica que el intérprete está esperando una orden.

Con el fin de verificar que todo ha sido instalado satisfactoriamente, se realizará una pequeña prueba. Abrir la interfaz de mongo, ejecutando el comando *mongo* en la linea de comandos de Debian. Una vez dentro, escribir los siguientes comandos:

```
> db.test.save( { a: 1 } )

> db.test.find()
```

Si el resultado es parecido a *{ "_id" : ObjectId("..."), "a" : 1 }* significa que todo ha sido instalado correctamente.

7.2 Importación de los datos

Con el objetivo de evaluar el desempeño del diseño *Un documento diario por punto de monitoreo*, se ha escrito una aplicación[3] en Java que importa los datos correspondientes a los meses de Septiembre, Octubre y Noviembre del 2012, tratados en detalle en la sección *5.8.1 Cifras sobre lo almacenado*.

Ilustración 42: Diagrama de flujo de los datos importados

3 Aplicación disponible en el repositorio: https://github.com/kiluax/ALMACurrentDesignToOneDocumentDesign

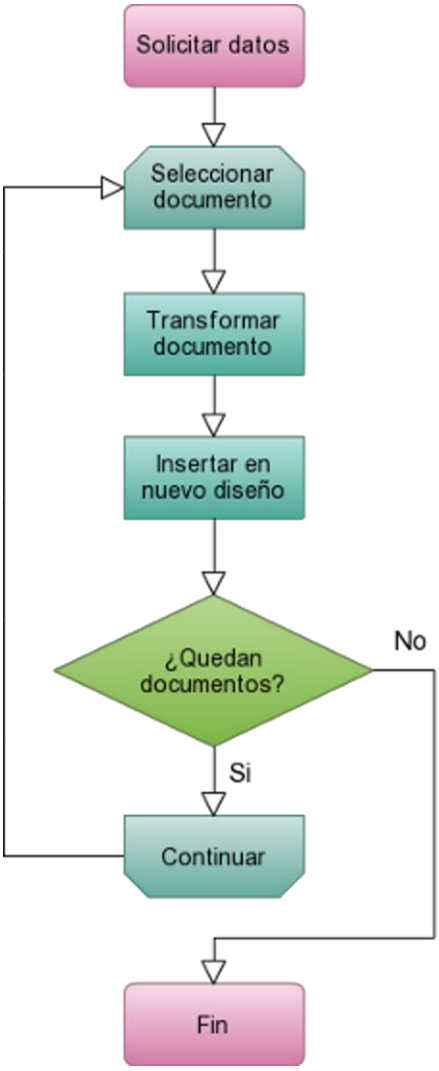


Ilustración 43: Diagrama de flujo de la aplicación de importación

7.3 Generador de datos

Durante el desarrollo de este estudio de factibilidad también se creó un generador de datos, que se encuentra disponible en el mismo repositorio que la aplicación de importación[3], pero bajo el nombre de "OneDocumentPerformanceTest".

Las primeras pruebas de rendimiento sobre el esquema propuesto fueron realizadas utilizando el generador de datos, el cual inserta valores simulados para los documentos pre-localizados. El generador no simula el flujo de muestras proveniente de ActiveMQ, sino más bien los datos necesarios para pre-localizar un documento. La Tabla 8 contiene los campos simulados.

Campo simulado	Valor
Antena	Antenna_[i]
Componente	Component_[j]
Punto de monitoreo	MP_[k]
Valor del punto de monitoreo	$na_1...a_n$

Tabla 8: Nomenclatura de valores simulados

Explicación Tabla 8:

i: Número de la antena simulada, 1 <= i <= 70

j: Número del componente simulado, 1 <= j <= 41

k: Número del punto de monitoreo simulado, 1 <= k <= 35

n: Cantidad de dígitos del escalar de una muestra, 1 <= n <= 7. Si el escalar de una muestra es de 5 dígitos, entonces el valor simulado será "naaaa". En cambio si el número de dígitos es 1 o 2, entonces el valor simulado será "na". Este valor no es generado por el generador de datos, sino que es parte de la clase MongoManager de la aplicación de importación y corresponde a un valor estándar. El simular sólo entrega el número de dígitos y la clase MongoManager se encarga de fabricar el valor.

8 PRUEBAS

8.1 Introducción

Este capítulo es decisivo en determinar si el diseño *"Un documento diario por punto de monitoreo"* es capaz de cumplir con los requerimientos del TMC.

En primer lugar, se presenta el entorno hardware y software en el cual se realizan las pruebas, ésto es necesario para conocer la potencia de las máquinas involucradas.

Luego, se especifican los conjuntos de pruebas utilizados para evaluar tanto el diseño actual como el propuesto y los resultados obtenidos son presentados en gráficos para facilitar su lectura.

Finalmente, se realiza un análisis estadístico sobre los rendimientos obtenidos en ambos diseños y se revisa el funcionamiento del índice y el sharding para el diseño propuesto.

8.2 Entorno hardware y software

El cluster de pruebas de MongoDB está compuesto por dos nodos de datos, *mongo-r1* y *mongo-r2*, ver Ilustración 44, dos servidores de enrutamiento (Mongo routers) y tres de configuración (Mongo config). La Tabla 9 presenta todas las máquinas involucradas en el desarrollo de este estudio.

Máquina	CPU	RAM	Sistema Operativo
mongo-r1.osf.alma.cl	Intel Xeon X5355 2,66 GHz	16 GB	CentOS 5.8 64 bits
mongo-r2.osf.alma.cl	Intel Xeon X5355 2,66 GHz	16 GB	CentOS 5.8 64 bits
doris.osf.alma.cl	Intel Xeon E5-2650 2,0 GHz x 2	2.000 MB	Red Hat Enterprise Linux Server Release 6.4 64 bits
Local	AMD Turion64 2,2 GHz	1.536 MB	Debian 6.0.7 64 bits
3 Servidores de configuración MongoDB	1 CPU, máquina virtual	512 MB	CentOS 5.8 64 bits
2 Servidores de enrutamiento MongoDB	1 CPU, máquina virtual	1.024 MB	CentOS 5.8 64 bits

Tabla 9: Información de los servidores involucrados en las pruebas

Los servidores de enrutamiento son utilizados por MongoDB para direccionar las sentencias de selección e inserción y también proveen de un sistema de balanceo de carga, entre otras funcionalidades.

Los servidores de configuración son utilizados por MongoDB para mantener meta-información sobre lo que almacenan los nodos de datos del cluster y proveer de una fuente de rápido acceso para el direccionamiento de las sentencias.

La Tabla 9 también contiene las características del servidor en el cual se ejecutan las aplicaciones y pruebas que envían y obtienen datos de MongoDB (doris.osf.alma.cl). A modo de documentación, se incluyen las características de la máquina en la que se realizaron las primeras pruebas, de nombre "local", que es el computador portátil del autor de este estudio.

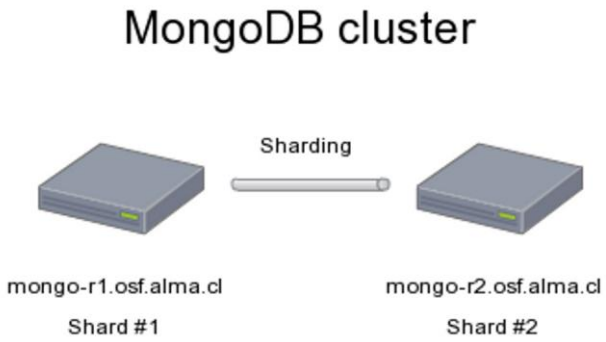


Ilustración 44: Cluster de pruebas MongoDB

8.3 Recordando el cache de MongoDB

Una de los aspectos fundamentales a la hora de realizar pruebas de rendimiento, es considerar todas aquellas variables que puedan afectar, tanto positiva como negativamente, en el resultado final. El caché interno de MongoDB es una de ellas. La sección *4.2.9 Caché de MongoDB*, cubre en más detalle este comportamiento.

Las consultas que se verán en las próximos tópicos, han sido formuladas tomando en cuenta esta funcionalidad. Las medidas precautorias son:

1. Formular tres conjuntos distintos de consultas de selección para cada diseño. Refiérase a distintos como: antenas, componentes y puntos de monitoreo.
2. Fechas distintas entre consultas: Debido a que las consultas de un conjunto son ejecutado en serie, debe existir un periodo de tiempo razonablemente distante entre las fechas solicitadas de una consulta y otra.

Las medidas presentadas, fuerzan a la base de datos a realizar operaciones de lectura en disco dentro de las primeras tres pruebas. Debido a que los conjuntos de selección son ejecutados en serie, los datos correspondientes a las pruebas P01, P02 y P03 son ingresados a caché, provocando que en su próxima consulta el tiempo de extracción sea menor para muchos de ellos (P04, P05, ..., P14). El caché es limitado y a medida que transcurren las pruebas, MongoDB quitará de él ciertos datos para agregar otros. Este comportamiento hace que ciertos datos que ya han sido consultados no se encuentren en el caché y deban ser leídos desde el disco, aumentando el tiempo de consulta.

8.4 Especificación de las pruebas

El objetivo de estas pruebas es averiguar el rendimiento del diseño actual y del diseño propuesto, para posteriormente realizar un análisis comparativo entre ambos enfoques y determinar si la propuesta cumple o no los requerimientos del TMC.

8.4.1 Dominio de datos

8.4.1.1 Diseño actual

Con el fin de evaluar las consultas bajo condiciones idénticas, todas las pruebas realizadas sobre el diseño actual utilizan el mismo dominio de datos, el cual es descrito en la sección *5.8.1 Cifras sobre lo almacenado*.

8.4.1.2 Diseño propuesto

En un principio se pensó en probar el rendimiento del esquema propuesto utilizando los datos de la sección *5.8.1 Cifras sobre los almacenado*, para ello se creó una aplicación de importación de datos, vista en la sección *7.2 Importación de los datos*, que transforma los datos desde el formato actual al formato propuesto, sin embargo el tiempo que tomó la operación de importación fue bastante (más del que se disponía para ello) debido a la cantidad de datos que era necesario importar. Finalmente, se utilizó el generador de datos para crear el dominio de datos en el cual el diseño actual es probado. La Tabla 10 contiene la información de los datos insertados.

Mes	2 (Febrero)
Número de días	11
Tamaña promedio por documento	1,3 MB
Tamaño de la colección	1.375,23 GB
Número de antenas	70
Número de componentes por antena	41
Número de puntos de monitoreo por componente	35
Total documentos	1.104.950

Tabla 10: Dominio de datos para el diseño propuesto

Todos los documentos son insertados utilizando la técnica de pre-localización con tiempo de muestreo fijo, esto quiere decir que cada documento posee todos los segundos del día, lo cual es equivalente al documento de un punto de monitoreo con tiempo de muestreo de un segundo.

8.4.2 Consultas de selección

Se han creado seis conjuntos de consultas de selección, con el fin de determinar el tiempo de extracción para una variable.

Tres conjuntos pertenecen al diseño actual y los otros tres al diseño propuesto. Esto debe ser así, ya que los tiempos de extracción del diseño propuesto no son compatibles con los del diseño actual. Por ejemplo, en el diseño actual es posible consultar por rangos como 19:08:33 hasta 19:10:22, sin embargo debido a la forma en que se estructuran los datos en el diseño propuesto la consulta anterior es tediosa y difícil de realizar. En cambio las consultas con granularidad de un segundo, un minuto, horas o días completos son simples de realizar.

8.4.2.1 Conjunto S01

La Tabla 11 presenta las consultas realizadas para el siguiente componente, dispositivo y punto de monitoreo:

Componente: CONTROL/DV10/LLC

Dispositivo: P_DET

Punto de monitoreo: P_DET

Tiempo de muestreo: 1 segundo

Consulta	Desde	Hasta	Tiempo (hrs.)	N° Elementos
S01-1	2012-11-14 22:34:00	2012-11-14 22:44:00	10/60	0
S01-2	2012-10-12 23:40:00	2012-10-13 00:10:00	30/60	0
S01-3	2012-09-10 16:15:00	2012-09-10 17:15:00	1	358
S01-4	2012-11-10 00:30:00	2012-11-10 03:30:00	3	40
S01-5	2012-11-20 14:01:00	2012-11-20 20:01:00	6	108
S01-6	2012-09-25 08:00:00	2012-09-25 20:00:00	12	3.124
S01-7	2012-10-05 10:30:00	2012-10-06 10:30:00	24	3.860

Tabla 11: Conjunto de consultas S01 del esquema actual

8.4.2.2 Conjunto S02

La Tabla 12 presenta las consultas realizadas para el siguiente componente, dispositivo y punto de monitoreo:

Componente: CONTROL/DV06/LLC

Dispositivo: POL_MON4

Punto de monitoreo: POL_MON4

Tiempo de muestreo: 1 segundo

Consulta	Desde	Hasta	Tiempo (hrs.)	N° Elementos
S02-1	2012-10-14 12:20:00	2012-10-14 12:30:00	10/60	40
S02-2	2012-09-25 05:40:00	2012-09-25 06:10:00	30/60	0
S02-3	2012-11-10 19:13:00	2012-11-10 20:13:00	1	20
S02-4	2012-11-19 23:30:00	2012-11-20 02:30:00	3	20
S02-5	2012-10-03 14:01:00	2012-10-03 21:01:00	6	440
S02-6	2012-09-29 06:05:00	2012-09-29 18:05:00	12	7.230
S02-7	2012-09-29 01:00:00	2012-09-30 01:00:00	24	11.890

Tabla 12: Conjunto de consultas S02 del esquema actual

8.4.2.3 Conjunto S03

La Tabla 13 presenta las consultas realizadas para el siguiente componente, dispositivo y punto de monitoreo:

Componente: CONTROL/DV10/FrontEnd/Cryostat

Dispositivo: GATE_VALVE_STATE

Punto de monitoreo: GATE_VALVE_STATE

Tiempo de muestreo: 5 segundos

Consulta	Desde	Hasta	Tiempo (hrs.)	N° Elementos
S03-1	2012-09-18 09:20:00	2012-09-18 09:30:00	10/60	0
S03-2	2012-10-20 13:25:00	2012-10-20 13:55:00	30/60	0
S03-3	2012-10-05 11:00:00	2012-10-05 12:00:00	1	8
S03-4	2012-11-21 11:09:00	2012-11-21 14:09:00	3	8

S03-5	2012-09-25 23:45:00	2012-09-26 05:45:00	6	340
S03-6	2012-11-16 15:43:00	2012-11-17 03:43:00	12	389
S03-7	2012-09-09 01:50:00	2012-09-10 01:50:00	24	4.052

Tabla 13: Conjunto de consultas S03 del esquema actual

8.4.2.4 Conjunto Q01

La Tabla 14 presenta las consultas realizadas para la siguiente antena, componente y punto de monitoreo:

Antena: Antenna_1

Componente: Component_1

Punto de monitoreo: MP_1

Consulta	Desde	Hasta	Tiempo (hrs.)	N° Muestras
Q01-1	2013-2-2 19:32:15	-----	1/3600	1
Q01-2	2013-2-9 22:04:00	2013-2-9 22:04:59	1/60	60
Q01-3	2013-2-3 16:00:00	2013-2-3 16:59:59	1	3.600
Q01-4	2013-2-5 19:00:00	2013-2-5 21:59:59	3	10.800
Q01-5	2013-2-5 01:00:00	2013-2-5 06:59:59	6	21.600
Q01-6	2013-2-7 11:00:00	2013-2-7 22:59:59	12	43.200
Q01-7	2013-2-1 00:00:00	2012-2-1 23:59:59	24	86.400

Tabla 14: Conjunto de consultas Q01 del esquema propuesto

8.4.2.5 Conjunto Q02

La Tabla 15 presenta las consultas realizadas para la siguiente antena, componente y punto de monitoreo:

Antena: Antenna_6

Componente: Component_23

Punto de monitoreo: MP_19

Consulta	Desde	Hasta	Tiempo (hrs.)	N° Muestras
Q02-1	2013-2-8 15:58:59	-----	1/3600	1
Q02-2	2013-2-7 04:00:00	2013-2-7 04:00:59	1/60	60
Q02-3	2013-2-1 01:00:00	2013-2-1 01:59:59	1	3.600
Q02-4	2013-2-9 05:00:00	2013-2-9 07:59:59	3	10.800
Q02-5	2013-2-3 11:00:00	2013-2-3 16:59:59	6	21.600
Q02-6	2013-2-8 03:00:00	2013-2-8 14:59:59	12	43.200
Q02-7	2013-2-10 00:00:00	2013-2-10 23:59:59	24	86.400

Tabla 15: Conjunto de consultas Q02 del esquema propuesto

8.4.2.6 Conjunto Q03

La Tabla 16 presenta las consultas realizadas para la siguiente antena, componente y punto de monitoreo:

Antena: Antenna_5

Componente: Component_40

Punto de monitoreo: MP_34

Consulta	Desde	Hasta	Tiempo (hrs.)	N° Muestras
Q03-1	2013-2-6 13:28:09	-----	1/3600	1
Q03-2	2013-2-5 08:11:00	2013-2-5 08:11:59	1/60	60
Q03-3	2013-2-9 23:00:00	2013-2-9 23:59:59	1	3.600
Q03-4	2013-2-2 11:00:00	2013-2-2 13:59:59	3	10.800
Q03-5	2013-2-10 03:00:00	2013-2-10 08:59:59	6	21.600
Q03-6	2013-2-1 04:00:00	2013-2-1 15:59:59	12	43.200
Q03-7	2013-2-6 00:00:00	2013-2-6 00:00:00	24	86.400

Tabla 16: Conjunto de consultas Q03 del esquema propuesto

8.4.3 Conjunto de inserción

8.4.3.1 Conjunto I01

Se ha definido este conjunto de inserción con la finalidad de indicar aquellas pruebas en las que existe un flujo de inserción hacia la base de datos, para determinar si ello afecta o no al rendimiento de las consultas de selección.

En el caso de las pruebas sobre el esquema actual, el flujo de inserciones que se utiliza son datos reales provenientes del monitoreo de las antenas. Para realizar esto, sólo basta con iniciar el software TMC correspondiente a MongoDB, TMCMongoArchiver, para que los datos sean obtenidos desde la cola ActiveMQ y puestos en la base de datos.

Por otra parte, las inserciones del conjunto *I01* para el diseño propuesto provienen del generador de datos, el cual insertará datos ficticios en la base de datos correspondientes al día 11.

8.4.4 Conjunto P[n]

El conjunto *P[n]* o pruebas de rendimiento es la denominación utilizada para las pruebas de desempeño que aglomeran a uno o más conjuntos.

ID	Esquema Actual	Esquema Propuesto
P01	S01	Q01
P02	S02	Q02
P03	S03	Q03
P04	S01 y S02	Q01 y Q02
P05	S01 y S03	Q01 y Q03
P06	S02 y S03	Q02 y Q03
P07	S01, S02 y S3	Q01, Q02 y Q3
P08	I01 y S01	I01 y Q01
P09	I01 y S02	I01 y Q02
P10	I01 y S03	I01 y Q03
P11	I01, S01 y S02	I01, Q01 y Q02
P12	I01, S01 y S03	I01, Q01 y Q03
P13	I01, S02 y S03	I01, Q02 y Q03
P14	I01, S01, S02 y S03	I01, Q01, Q02 y Q03

Tabla 17: Especificación de pruebas del conjunto P[n]

Las pruebas donde se incluyen más de un conjunto son ejecutadas en paralelo, es decir, al mismo tiempo y en la misma base de datos. La técnica utilizada para paralelizar los conjuntos consiste en ejecutar cada uno en su propia terminal de linux.

MongoDB recibe consultas de tipo selección en las pruebas donde está presente el conjunto *S[n] y Q[n]*; de la misma forma, se realizan inserciones y selecciones en paralelo para las pruebas donde están presentes los conjuntos *I01* y *S[n] o Q[n]*.

El objetivo de ejecutar pruebas de selección cuando no existe flujo de inserciones hacia MongoDB, es evaluar y posteriormente comparar el rendimiento en ambos escenarios (con y sin flujo de escritura). Sin embargo, esto es sólo para motivos de análisis, ya que el contexto real en el cual funcionará la base de datos es: flujo elevado y constante de inserciones y en paralelo se efectuarían extracción de datos históricos.

8.5 Resultados

8.5.1 Esquema actual

8.5.1.1 Conjunto P01

La Ilustración 45 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P01*.

Query	Required time	Returned samples
S01-1	290,17	0
S01-2	4,52	0
S01-3	176,85	358
S01-4	416,25	40
S01-5	369,34	108
S01-6	350,27	3.124
S01-7	5,94	3.860

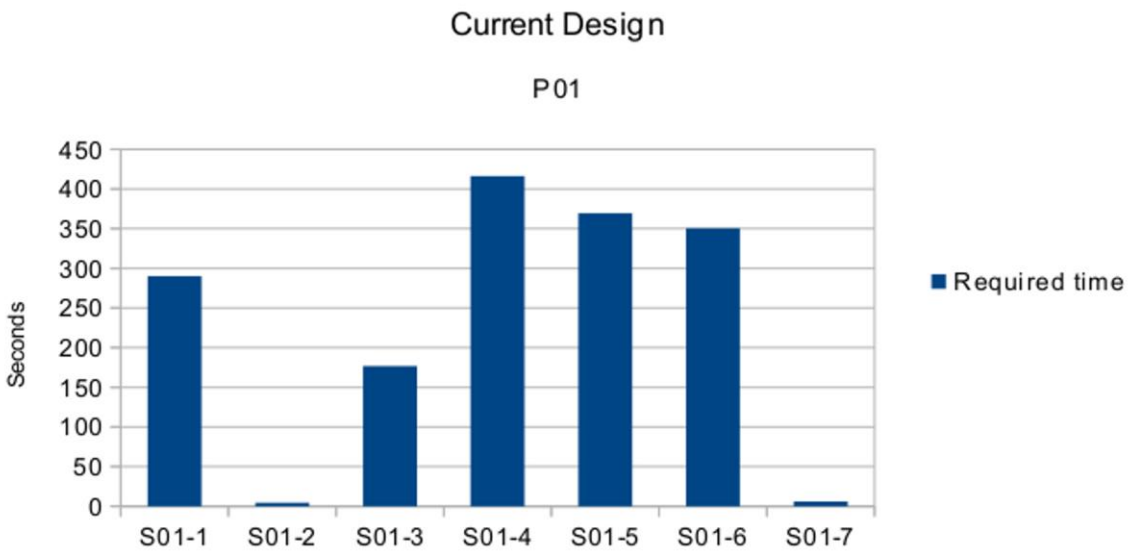


Ilustración 45: Resultados conjunto P01 - esquema actual

El menor tiempo de extracción requerido por S01-2 y S01-7 se debe a que poseen menos documentos indexados, lo que conlleva a una menor cantidad de entradas en sus índices y menos documentos por escanear en cada consulta, disminuyendo su tiempo de extracción.

8.5.1.2 Conjunto P02

La Ilustración 46 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P02*.

Query	Required time	Returned samples
S02-1	1,73	40
S02-2	286,04	0
S02-3	1,52	20
S02-4	1,45	20
S02-5	2,33	440
S02-6	282,18	7.230
S02-7	176,05	11.890

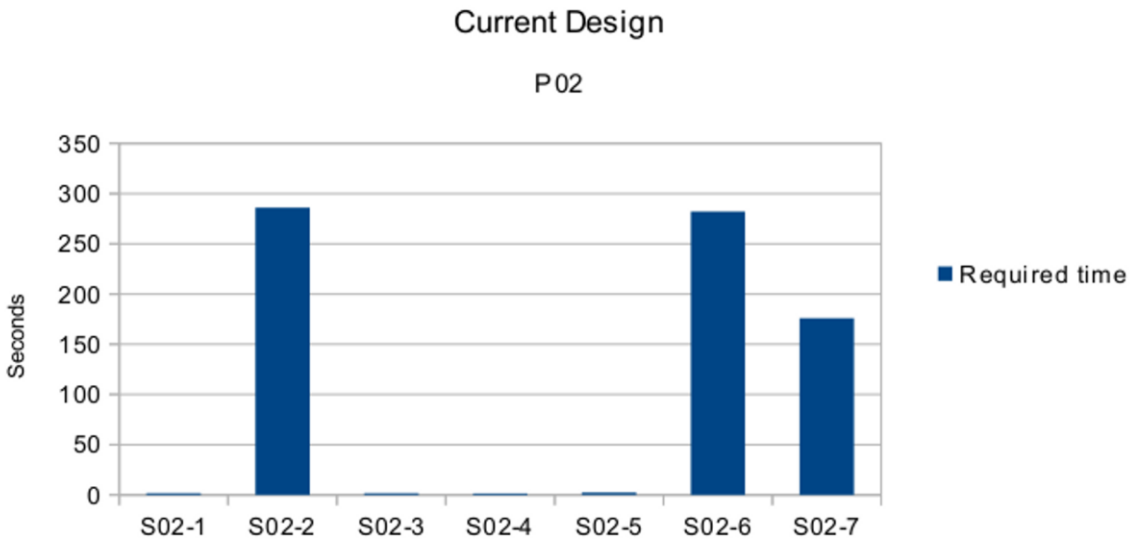


Ilustración 46: Resultados conjunto P02 - esquema actual

En este caso, MongoDB debió escanear varios documentos para determinar que en S02-2 ninguno cumplía la condición, ello explica su rendimiento deficiente.

8.5.1.3 Conjunto P03

La Ilustración 47 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P03*.

Query	Required time	Returned samples
S03-1	109,29	0
S03-2	39,9	0
S03-3	0,3	8
S03-4	0,3	8
S03-5	0,83	340
S03-6	1,08	389
S03-7	2,82	4.052

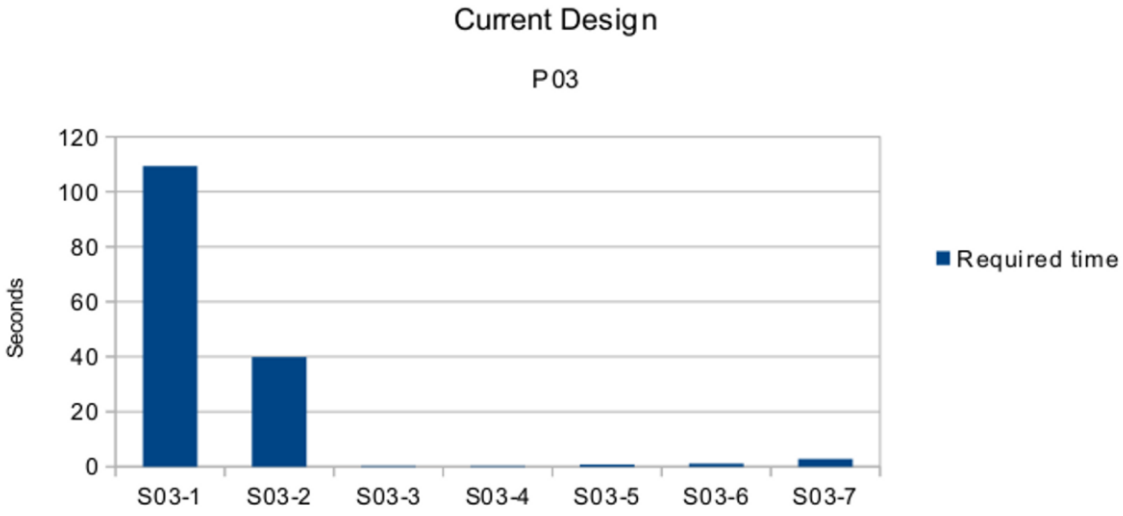


Ilustración 47: Resultados conjunto P03 - esquema actual

Al igual que el caso anterior, MongoDB debió escanear varios documentos para determinar que en S03-1 y S03-2 ninguno cumplía la condición, ello explica su rendimiento deficiente.

8.5.1.4 Conjunto P04

La Ilustración 48 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P04,* el cual corresponde a la ejecución de los conjuntos *S01* y *S02* en paralelo.

Query	S01	S02	Returned samples
P04-1	855,71	872,53	40
P04-2	350,17	307,89	0
P04-3	168,88	577,99	378
P04-4	995,87	763,87	60
P04-5	715,34	702,56	548
P04-6	661,06	906,08	10.354
P04-7	346,42	314,74	15.750

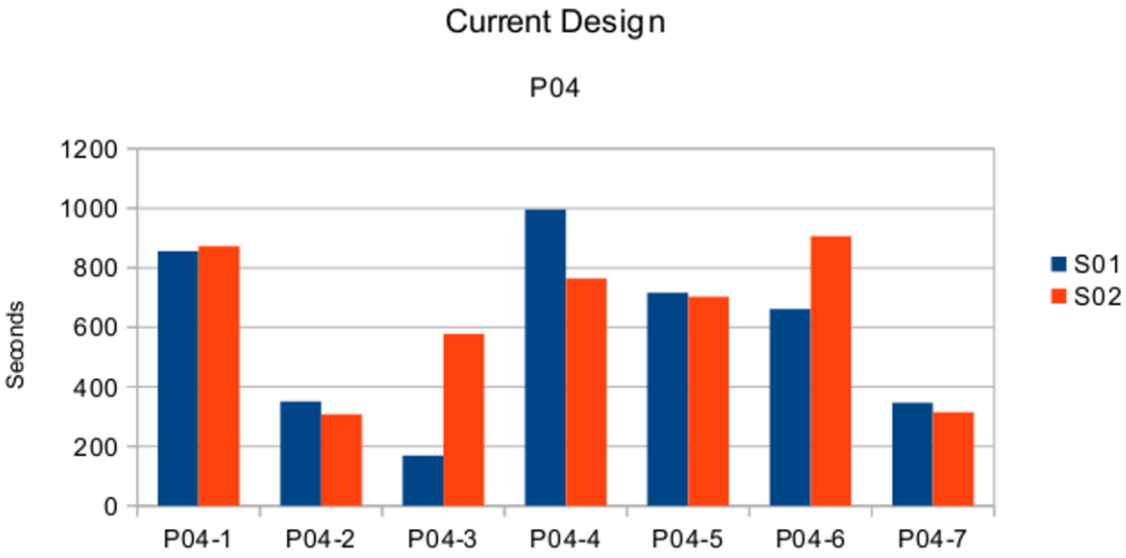


Ilustración 48: Resultados conjunto P04 - esquema actual

En términos generales, extraer una gran cantidad de documentos toma más tiempo que una pequeña. En este caso, ello no se aplica, ya que como muestra la Ilustración 48, P04-6 y P04-7 tienen el mayor número de datos y un tiempo de extracción mucho menor que varios de sus pares, ésto se debe, principalmente, al desempeño que tiene el índice que soporta dichas consultas.

8.5.1.5 Conjunto P05

La Ilustración 49 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P05,* el cual corresponde a la ejecución de los conjuntos *S01* y *S03* en paralelo.

Query	S01	S03	Returned samples
P05-1	413,67	187,55	0
P05-2	1,44	40,51	0
P05-3	81,32	0,31	366
P05-4	1,96	0,32	48
P05-5	1,61	0,9	448
P05-6	3,4	0,93	3.513
P05-7	6,57	2,6	7.912

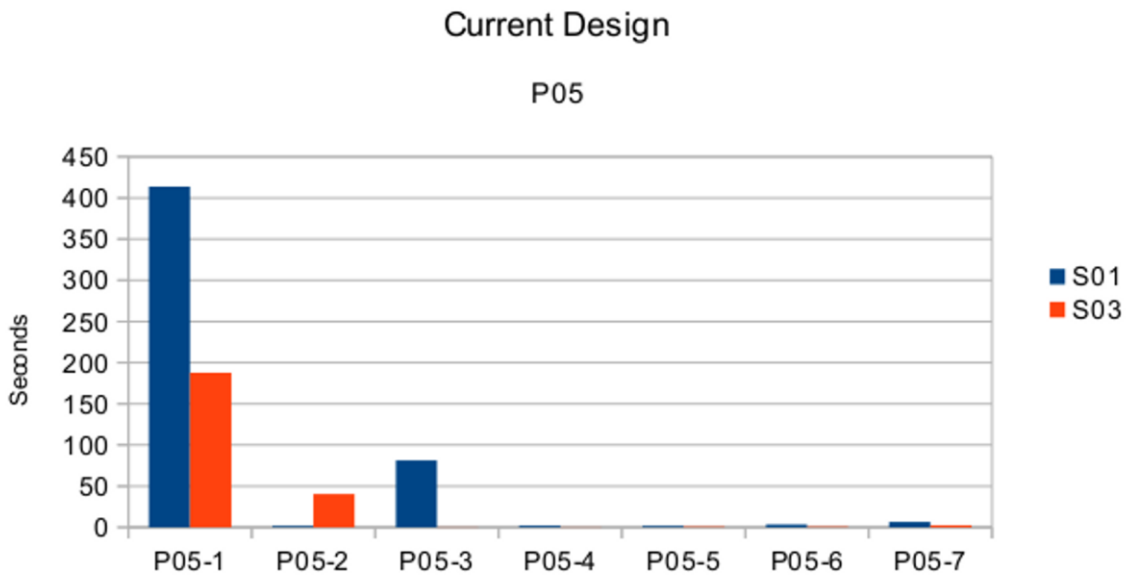


Ilustración 49: Resultados conjunto P05 - esquema actual

Al igual que en la sección *8.5.1.3 Conjunto P03*, MongoDB debió escanear varios documentos para determinar que en P05-1 y P05-2 ninguno cumplía la condición, ello explica su rendimiento deficiente.

8.5.1.6 Conjunto P06

La Ilustración 50 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P06,* el cual corresponde a la ejecución de los conjuntos *S02* y *S03* en paralelo.

Query	S02	S03	Returned samples
P06-1	280,23	174,25	40
P06-2	304,68	0,43	0
P06-3	1,87	0,37	28
P06-4	1,43	0,32	28
P06-5	1,87	0,8	780
P06-6	300,08	0,95	7.619
P06-7	304,98	3,22	15.942

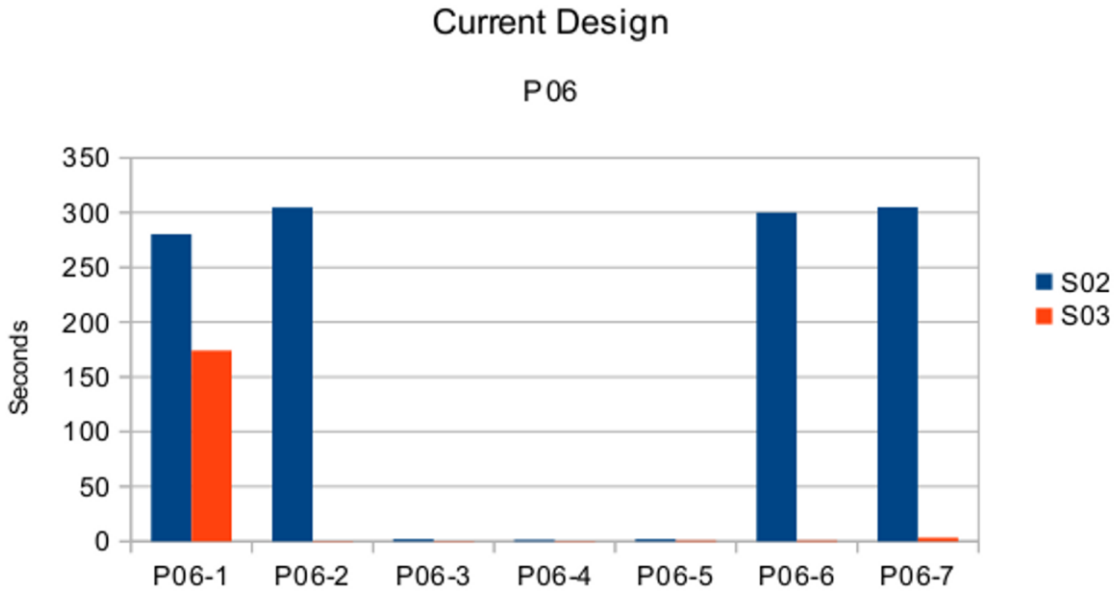


Ilustración 50: Resultados conjunto P06 - esquema actual

Este caso muestra el deficiente rendimiento de P06-1, P06-2, P06-6 y P06-7 que es arrastrado desde el conjunto S02, mismo caso que en la prueba de la sección *8.5.1.2 Conjunto P02*.

8.5.1.7 Conjunto P07

La Ilustración 51 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P07,* el cual corresponde a la ejecución de los conjuntos *S01, S02* y *S03* en paralelo.

Query	S01	S02	S03	Returned samples
P07-1	86,85	1,63	0,47	40
P07-2	1,44	449,13	71,24	0
P07-3	153,68	2,15	0,3	386
P07-4	1,53	1,46	0,38	68
P07-5	1,53	1,64	1,01	888
P07-6	2,28	288,22	0,74	10.743
P07-7	11,2	294,98	2,89	19.802

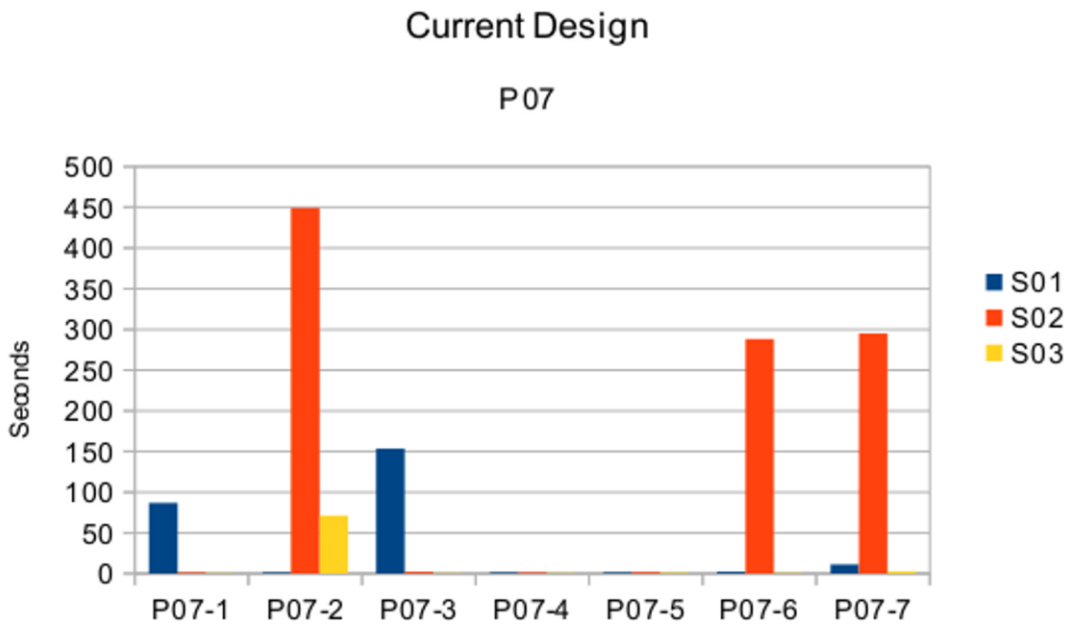


Ilustración 51: Resultados conjunto P07 - esquema actual

8.5.1.8 Conjunto P08

La Ilustración 52 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P08,* el cual está compuesto por un flujo de inserciones y uno de selección, *I01* y *S01* respectivamente.

Query	S01	Returned samples
P08-1	66,25	0
P08-2	4,02	0
P08-3	67,59	358
P08-4	11,54	40
P08-5	1,7	108
P08-6	2,75	3.124
P08-7	11,72	3.860

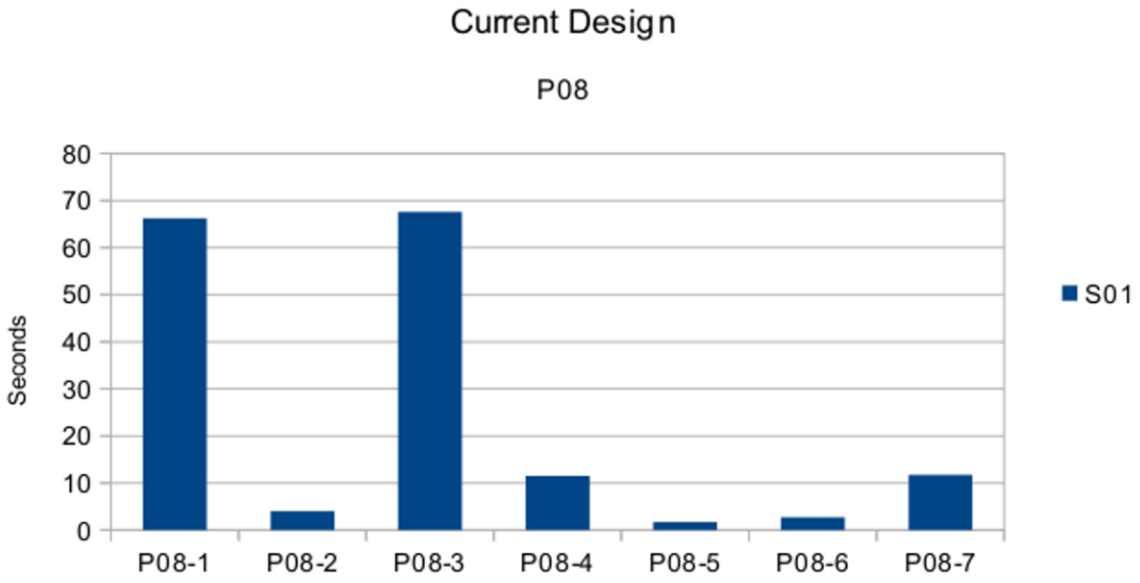


Ilustración 52: Resultados conjunto P08 - esquema actual

8.5.1.9 Conjunto P09

La Ilustración 53 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P09,* el cual está compuesto por un flujo de inserciones y uno de selección, *I01* y *S02* respectivamente.

Query	S02	Returned samples
P09-1	58,15	40
P09-2	267,52	0
P09-3	23,37	20
P09-4	6,18	20
P09-5	1,97	440
P09-6	7,01	7.230
P09-7	10,17	11.890

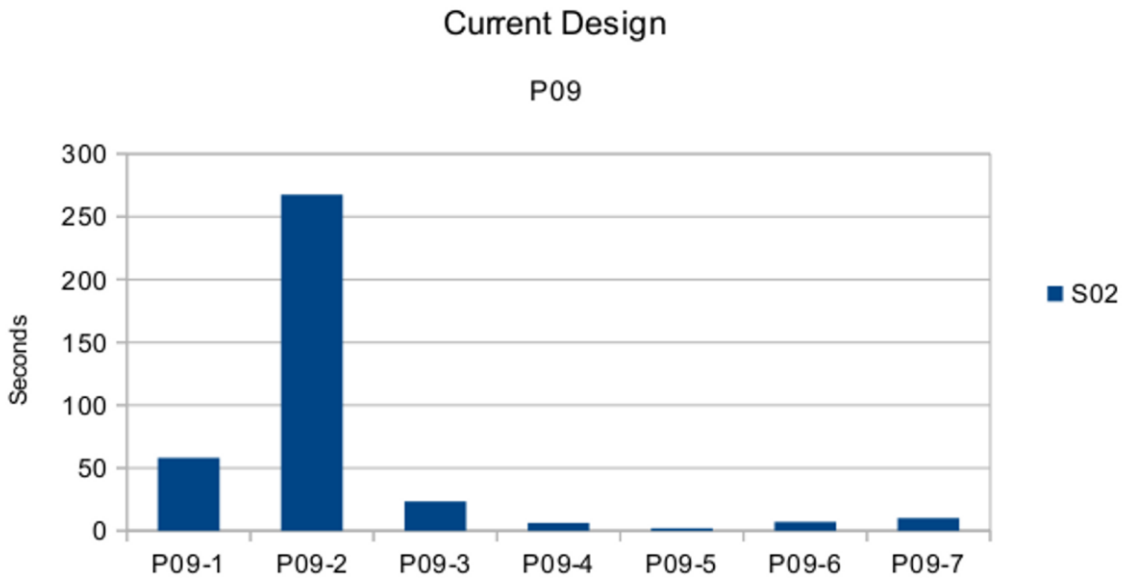


Ilustración 53: Resultados conjunto P09 - esquema actual

8.5.1.10 Conjunto P10

La Ilustración 54 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P10,* el cual está compuesto por un flujo de inserciones y uno de selección, *I01* y *S03* respectivamente.

Query	S03	Returned samples
P10-1	18,49	0
P10-2	39,5	0
P10-3	5,17	8
P10-4	0,43	8
P10-5	0,87	340
P10-6	2,31	389
P10-7	3,3	4.052

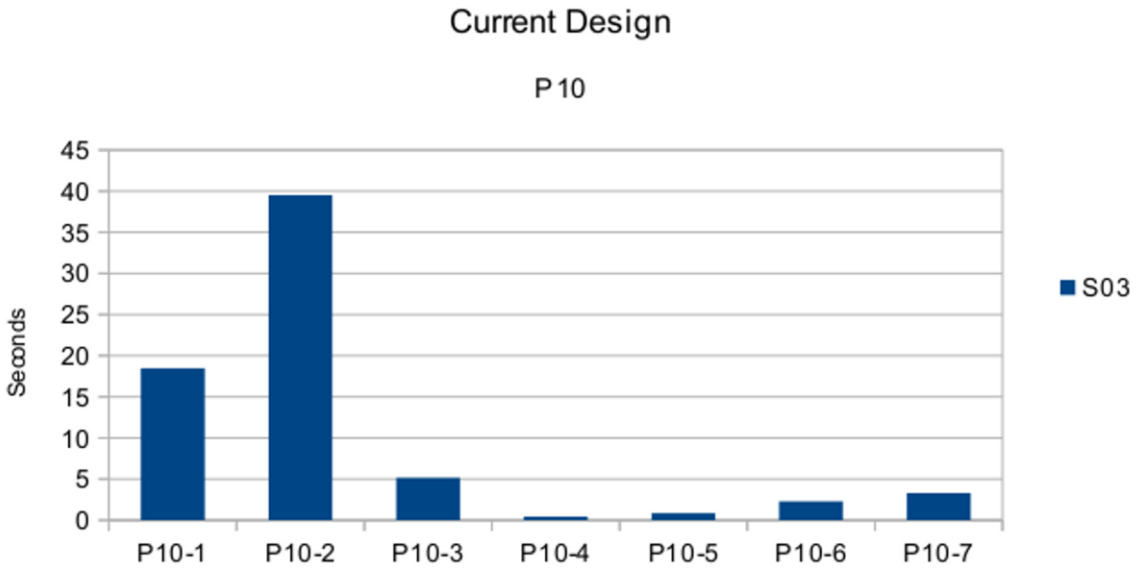


Ilustración 54: Resultados conjunto P10 - esquema actual

8.5.1.11 Conjunto P11

La Ilustración 55 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P11,* el cual está compuesto por un flujo de inserciones y dos de selección, *I01*, *S01* y *S02* respectivamente.

Query	S01	S02	Returned samples
P11-1	1,87	1,34	40
P11-2	4,39	1,56	0
P11-3	65,85	3,77	378
P11-4	3,5	3,05	60
P11-5	3,46	1,81	548
P11-6	2,37	5,21	10.354
P11-7	6,07	9,81	15.750

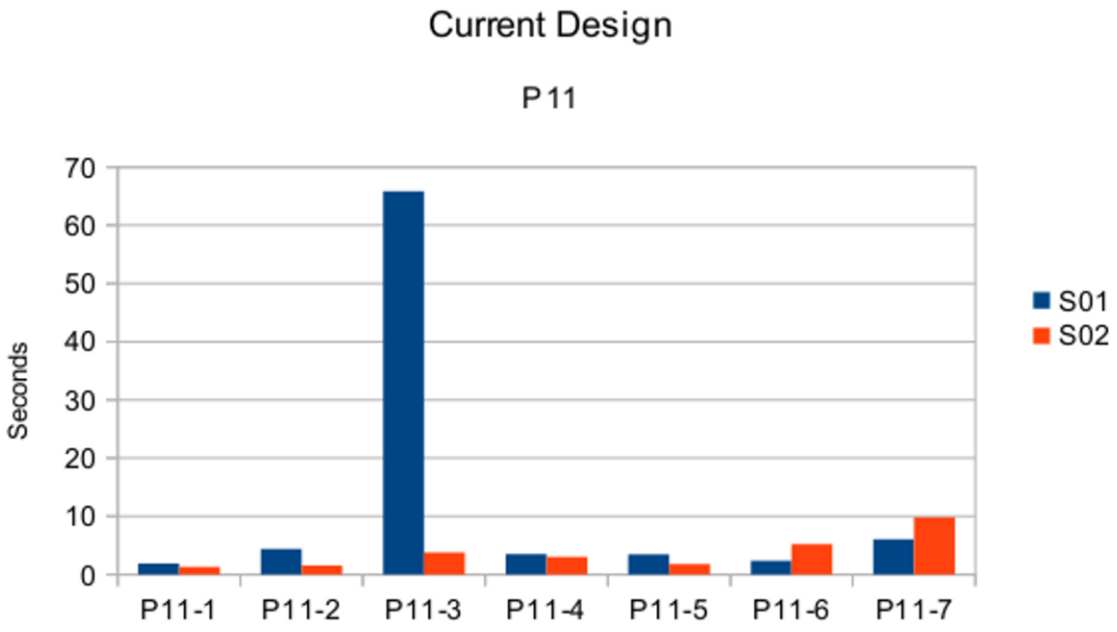


Ilustración 55: Resultados conjunto P11 - esquema actual

8.5.1.12 Conjunto P12

La Ilustración 56 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P12,* el cual está compuesto por un flujo de inserciones y dos de selección, *I01*, *S01* y *S03* respectivamente.

Query	S01	S03	Returned samples
P12-1	1,52	0,5	0
P12-2	3,42	0,43	0
P12-3	2,63	0,48	366
P12-4	3,87	0,37	48
P12-5	1,6	0,77	448
P12-6	2,23	0,7	3.513
P12-7	5,41	2,07	7.912

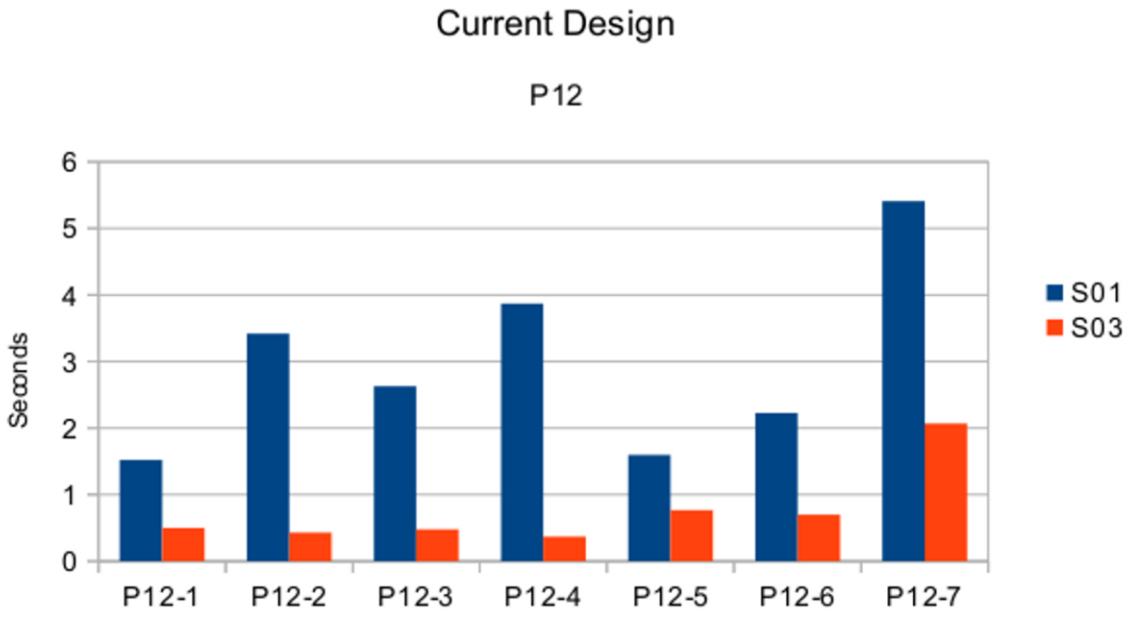


Ilustración 56: Resultados conjunto P12 - esquema actual

8.5.1.13 Conjunto P13

La Ilustración 57 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P13,* el cual está compuesto por un flujo de inserciones y dos de selección, *I01, S02 y S03* respectivamente.

Query	S02	S03	Returned samples
P13-1	2,72	0,52	40
P13-2	1,53	0,44	0
P13-3	3,27	0,43	28
P13-4	1,5	0,3	28
P13-5	1,86	0,84	780
P13-6	5,52	0,77	7.619
P13-7	7,71	2,17	15.942

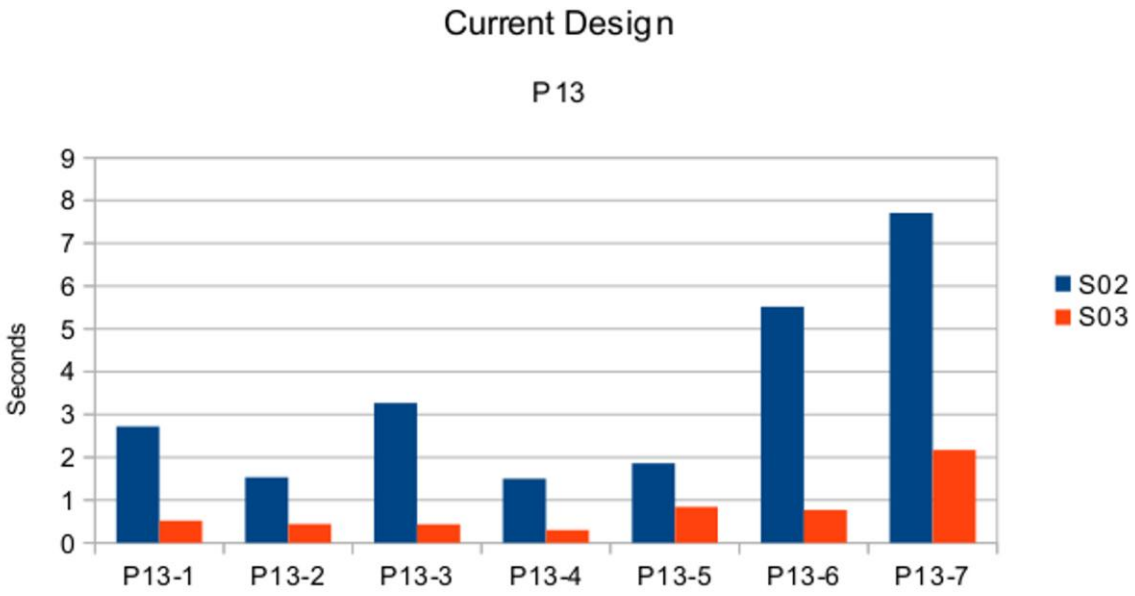


Ilustración 57: Resultados conjunto P13 - esquema actual

8.5.1.14 Conjunto P14

La Ilustración 58 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P14,* el cual está compuesto por un flujo de inserciones y tres de selección, *I01, S01, S02 y S03* respectivamente.

Query	S01	S02	S03	Returned samples
P14-1	1,49	1,11	9,67	40
P14-2	4,63	1,5	0,5	0
P14-3	2,02	6,12	0,55	386
P14-4	2,04	1,84	0,52	68
P14-5	2,97	2,13	0,85	888
P14-6	2,72	7,41	0,82	10.743
P14-7	6,58	14,26	2,5	19.802

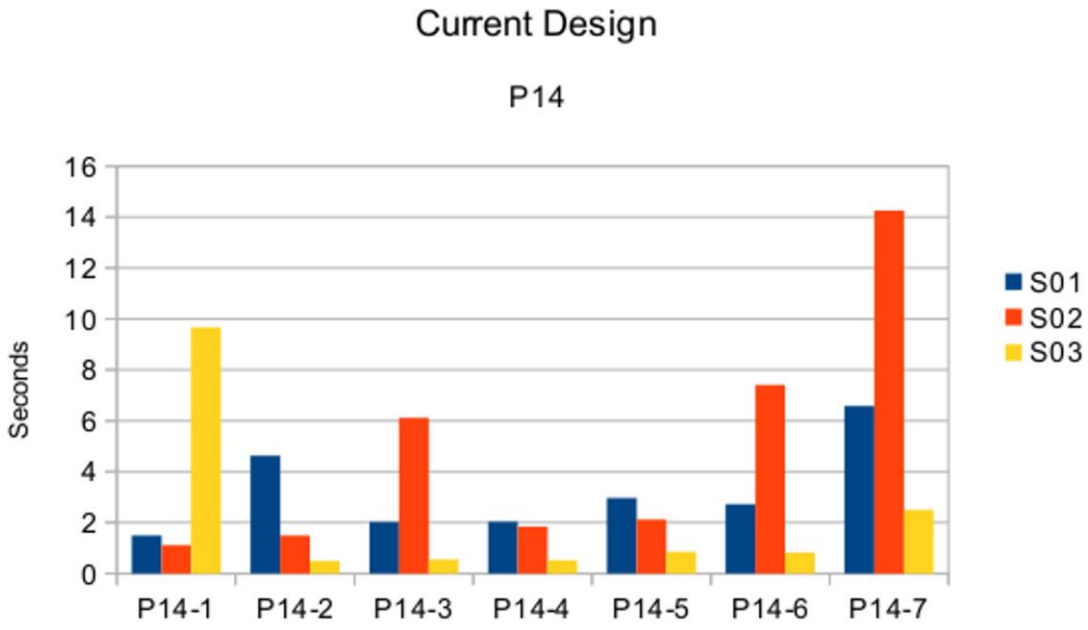


Ilustración 58: Resultados conjunto P14 - esquema actual

8.5.1.15 Análisis de desempeño del conjunto S01

La Ilustración 59 presenta una comparación en el rendimiento de todas las pruebas en las que está presente el conjunto de extracción *S01*.

Query	P01	P04	P05	P07	P08	P11	P12	P14	Returned samples
S01-1	290,17	855,71	413,67	86,85	66,25	1,87	1,52	1,49	0
S01-2	4,52	350,17	1,44	1,44	4,02	4,39	3,42	4,63	0
S01-3	176,85	168,88	81,32	153,68	67,59	65,85	2,63	2,02	358
S01-4	416,25	995,87	1,96	1,53	11,54	3,5	3,87	2,04	40
S01-5	369,34	715,34	1,61	1,53	1,7	3,46	1,6	2,97	108
S01-6	350,27	661,06	3,4	2,28	2,75	2,37	2,23	2,72	3.124
S01-7	5,94	346,42	6,57	11,2	11,72	6,07	5,41	6,58	3.860

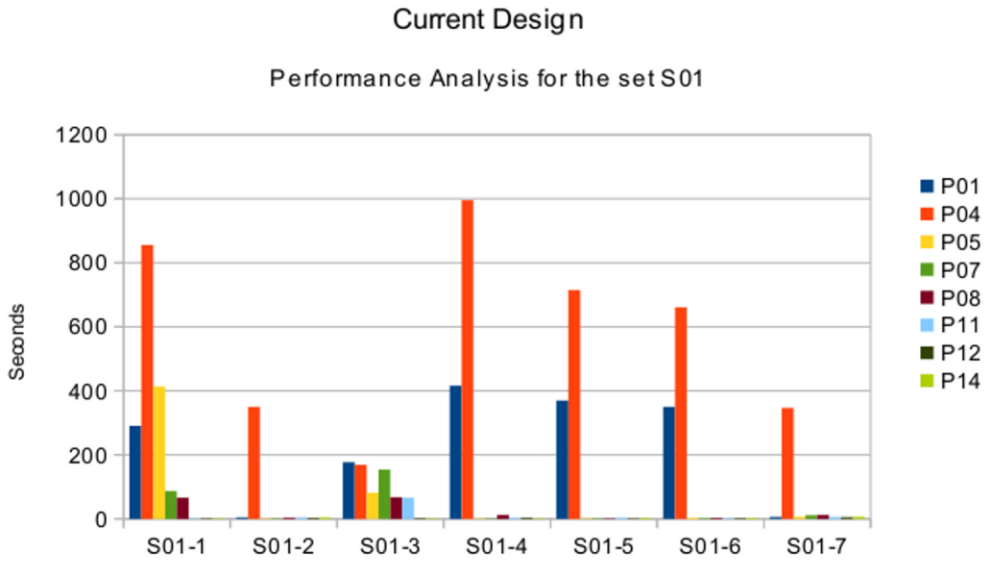


Ilustración 59: Análisis de rendimiento para el conjunto S01 del esquema actual

8.5.1.16 Análisis de desempeño del conjunto S02

La Ilustración 60 presenta una comparación en el rendimiento de todas las pruebas en las que está presente el conjunto de extracción *S02*.

Query	P02	P04	P06	P07	P09	P11	P13	P14	Returned samples
S02-1	1,73	872,53	280,23	1,63	58,15	1,34	2,72	1,11	40
S02-2	286,04	307,89	304,68	449,13	267,52	1,56	1,53	1,5	0
S02-3	1,52	577,99	1,87	2,15	23,37	3,77	3,27	6,12	20
S02-4	1,45	763,87	1,43	1,46	6,18	3,05	1,5	1,84	20
S02-5	2,33	702,56	1,87	1,64	1,97	1,81	1,86	2,13	440
S02-6	282,18	906,08	300,08	288,22	7,01	5,21	5,52	7,41	7.230
S02-7	176,05	314,74	304,98	294,98	10,17	9,81	7,71	14,26	11.890

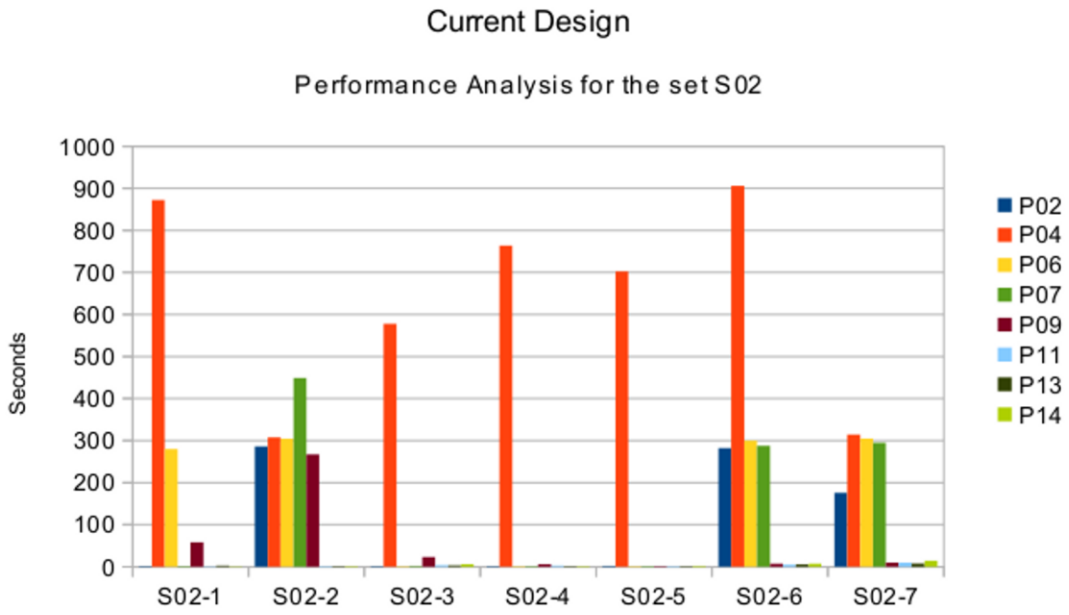


Ilustración 60: Análisis de rendimiento para el conjunto S02 del esquema actual

8.5.1.17 Análisis de desempeño del conjunto S03

La Ilustración 61 presenta una comparación en el rendimiento de todas las pruebas en las que está presente el conjunto de extracción *S03*.

Query	P03	P05	P06	P07	P10	P12	P13	P14	Returned samples
S03-1	109,29	187,55	174,25	0,47	18,49	0,5	0,52	9,67	0
S03-2	39,9	40,51	0,43	71,24	39,5	0,43	0,44	0,5	0
S03-3	0,3	0,31	0,37	0,3	5,17	0,48	0,43	0,55	8
S03-4	0,3	0,32	0,32	0,38	0,43	0,37	0,3	0,52	8
S03-5	0,83	0,9	0,8	1,01	0,87	0,77	0,84	0,85	340
S03-6	1,08	0,93	0,95	0,74	2,31	0,7	0,77	0,82	389
S03-7	2,82	2,6	3,22	2,89	3,3	2,07	2,17	2,5	4.052

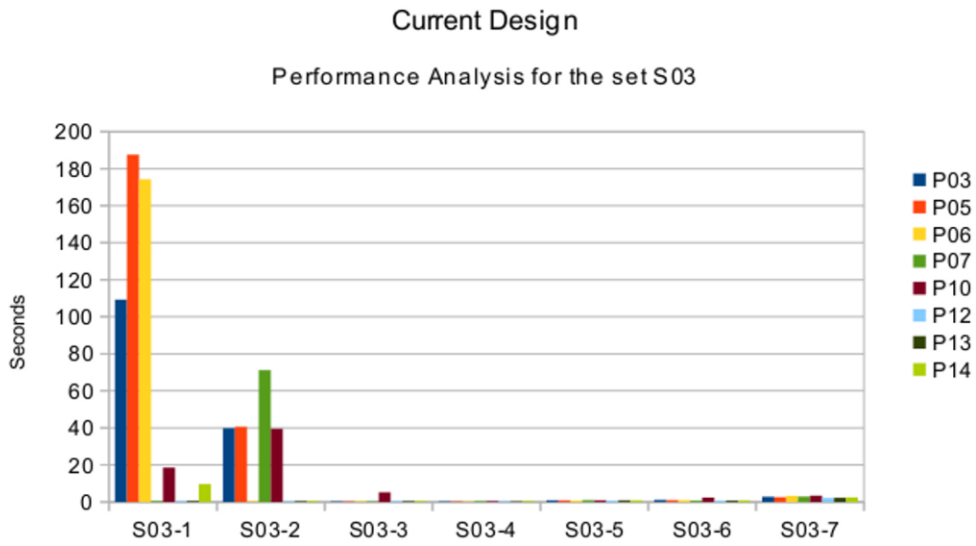


Ilustración 61: Análisis de rendimiento para el conjunto S03 del esquema actual

8.5.2 Un documento diario por punto de monitoreo

8.5.2.1 Conjunto P01

La Ilustración 62 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P01*.

Query	Required time	Returned samples
Q01-1	28	1
Q01-2	22	60
Q01-3	0	3.600
Q01-4	1	10.800
Q01-5	1	21.600
Q01-6	1	43.200
Q01-7	1	86.400

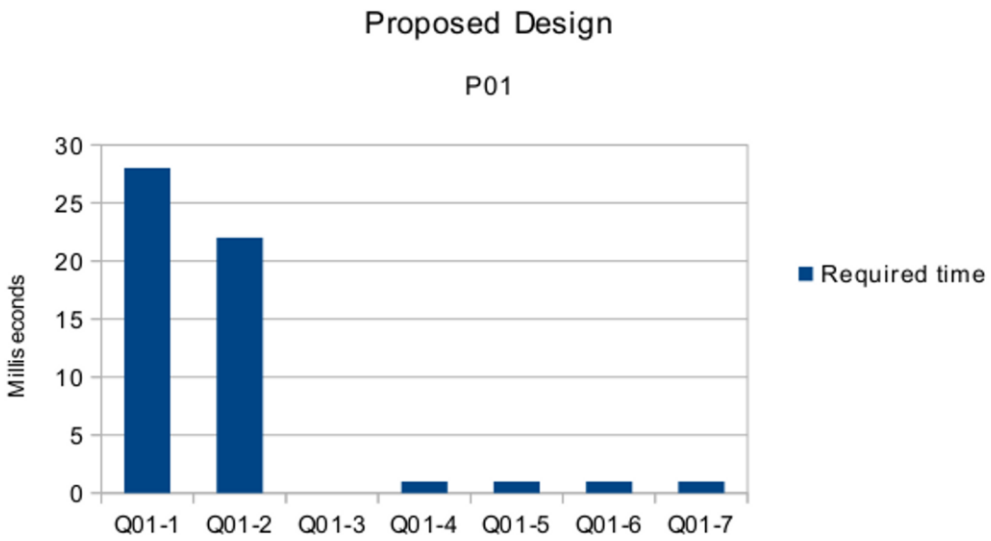


Ilustración 62: Resultados conjunto P01 - esquema propuesto

La anidación de documentos en Q01-1 tiene una pequeña penalidad de tiempo y a MongoDB le tomó unos pocos milisegundos más extraer una muestra de 1 segundo.

8.5.2.2 Conjunto P02

La Ilustración 63 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P02*.

Query	Required time	Returned samples
Q02-1	1	1
Q02-2	1	60
Q02-3	1	3.600
Q02-4	1	10.800
Q02-5	1	21.600
Q02-6	1	43.200
Q02-7	1	86.400

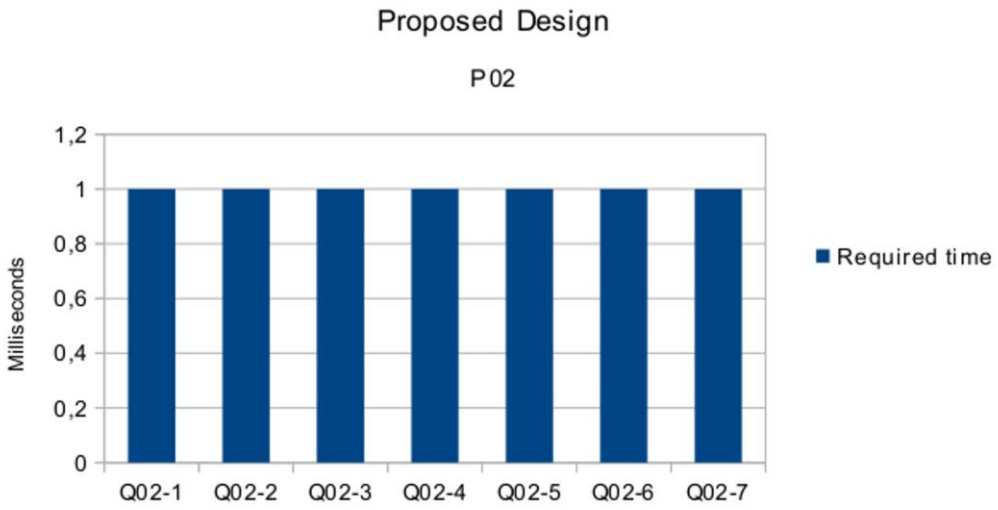


Ilustración 63: Resultados conjunto P02 - esquema propuesto

En este caso, MongoDB fue consistente en la extracción de las muestras de distinto tamaño, requiriendo sólo 1 milisegundo. Esto se debe a que las horas-minutos de los datos extraídos en el conjunto Q02 son en la mañana, lo cual significa consultas levemente más rápidas que en las últimas horas-minutos del día. Tema tratado en detalle en la sección *6.2.3 Un documento diario por punto de monitoreo*.

8.5.2.3 Conjunto P03

La Ilustración 64 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P03*.

Query	Required time	Returned samples
Q03-1	31	1
Q03-2	17	60
Q03-3	22	3.600
Q03-4	2	10.800
Q03-5	22	21.600
Q03-6	19	43.200
Q03-7	1	86.400

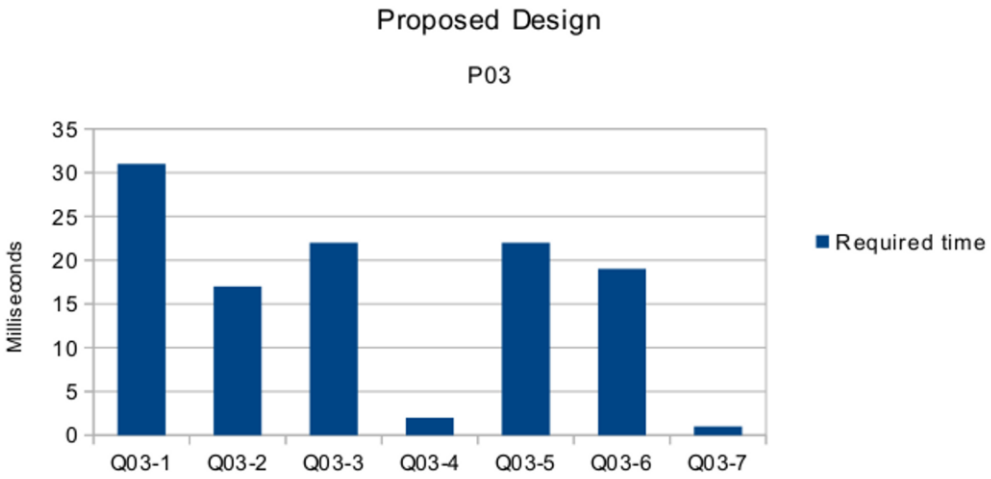


Ilustración 64: Resultados conjunto P03 - esquema propuesto

La extracción de Q03-7 es la más rápida del conjunto. Esto se debe a que el documento es extraído completamente y no es necesario recorrerlo en búsqueda de las muestras.

8.5.2.4 Conjunto P04

La Ilustración 65 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P04,* el cual corresponde a la ejecución de los conjuntos Q*01 y Q02* en paralelo.

Query	Q01	Q02	Returned samples
P04-1	1	1	2
P04-2	1	1	120
P04-3	0	0	7.200
P04-4	1	1	21.600
P04-5	1	1	43.200
P04-6	1	1	86.400
P04-7	1	0	172.800

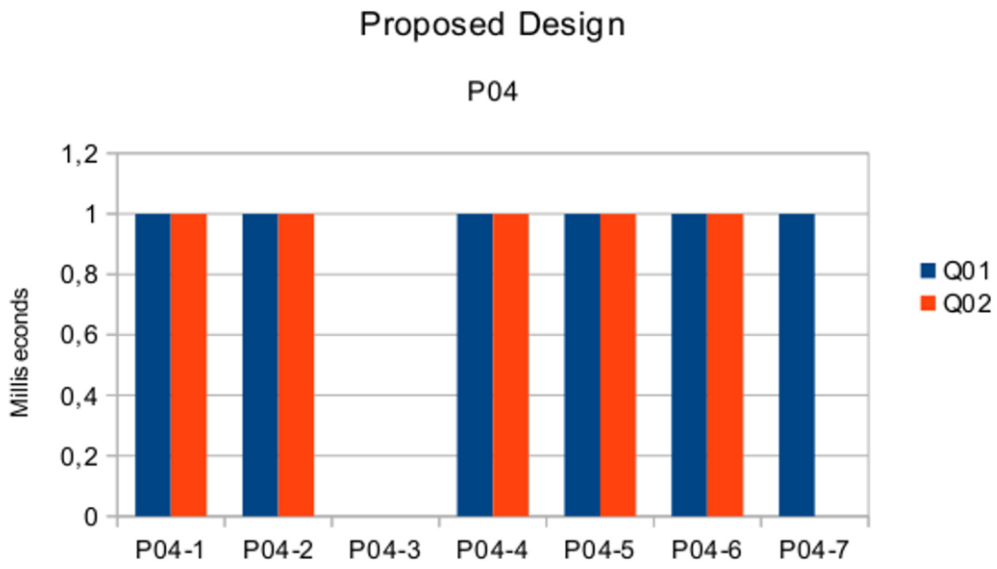


Ilustración 65: Resultados conjunto P04 - esquema propuesto

8.5.2.5 Conjunto P05

La Ilustración 66 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P05,* el cual corresponde a la ejecución de los conjuntos *Q01* y *Q03* en paralelo.

Query	Q01	Q03	Returned samples
P05-1	1	5	2
P05-2	1	0	120
P05-3	1	1	7.200
P05-4	1	1	21.600
P05-5	3	1	43.200
P05-6	0	0	86.400
P05-7	1	1	172.800

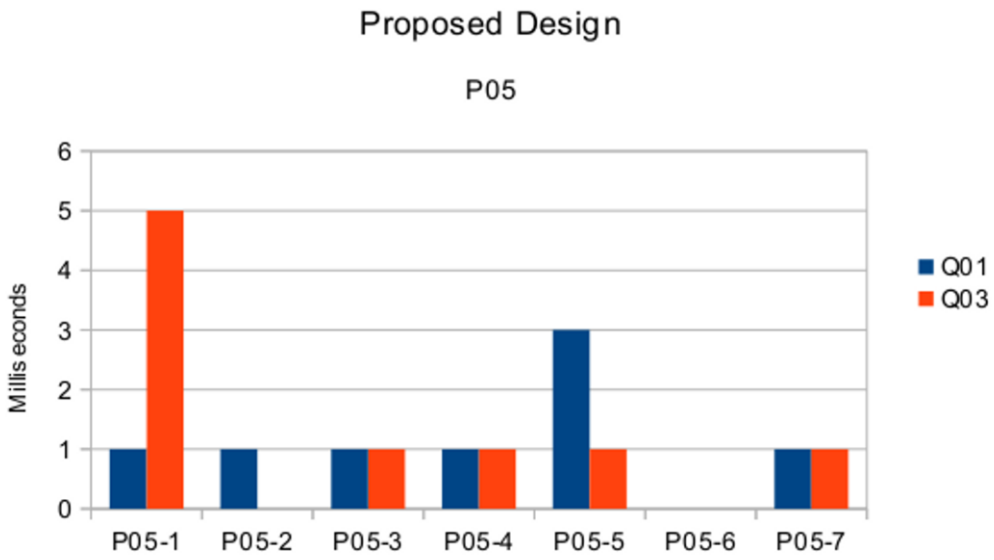


Ilustración 66: Resultados conjunto P05 - esquema propuesto

8.5.2.6 Conjunto P06

La Ilustración 67 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P06,* el cual corresponde a la ejecución de los conjuntos *Q02* y *Q03* en paralelo.

Query	Q02	Q03	Returned samples
P06-1	1	0	2
P06-2	1	1	120
P06-3	1	0	7.200
P06-4	1	0	21.600
P06-5	1	1	43.200
P06-6	0	0	86.400
P06-7	1	1	172.800

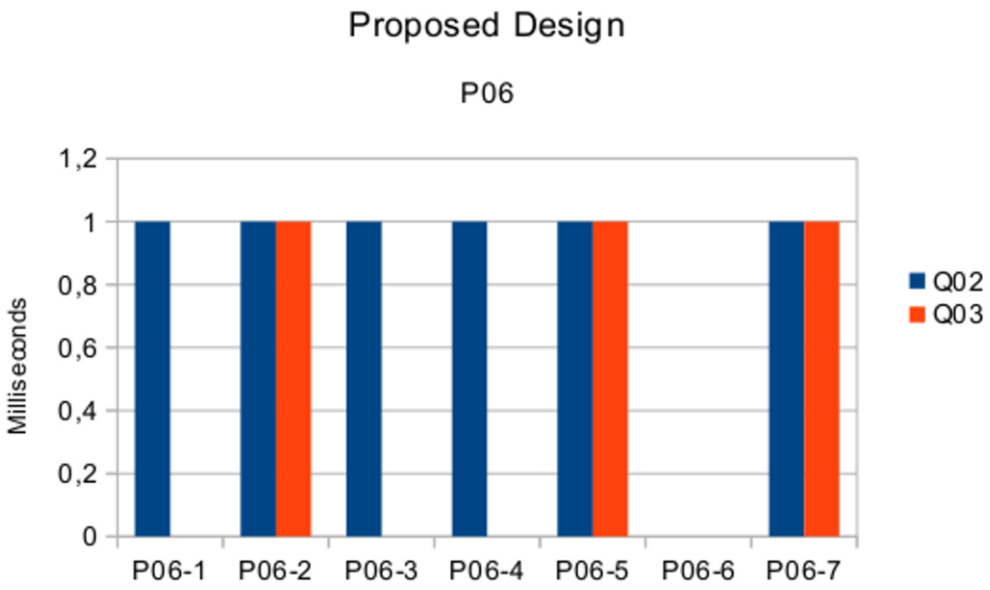


Ilustración 67: Resultados conjunto P06 - esquema propuesto

8.5.2.7 Conjunto P07

La Ilustración 68 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P07,* el cual corresponde a la ejecución de los conjuntos *Q01, Q02* y *Q03* en paralelo.

Query	Q01	Q02	Q03	Returned samples
P07-1	12	5	1	3
P07-2	1	1	1	180
P07-3	1	1	1	10.800
P07-4	1	1	0	32.400
P07-5	1	1	1	64.800
P07-6	1	1	1	129.600
P07-7	0	1	1	259.200

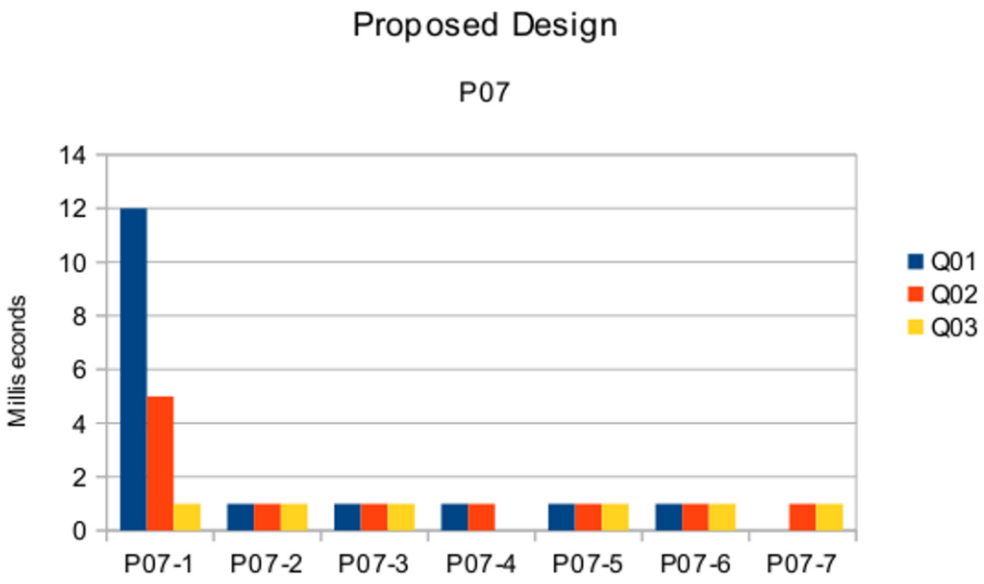


Ilustración 68: Resultados conjunto P07 - esquema propuesto

8.5.2.8 Conjunto P08

La Ilustración 69 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P08,* el cual corresponde a la ejecución de los conjuntos *I01* y *Q01* en paralelo.

Query	Q01	Returned samples
P08-1	1	1
P08-2	1	60
P08-3	1	3.600
P08-4	1	10.800
P08-5	1	21.600
P08-6	1	43.200
P08-7	1	86.400

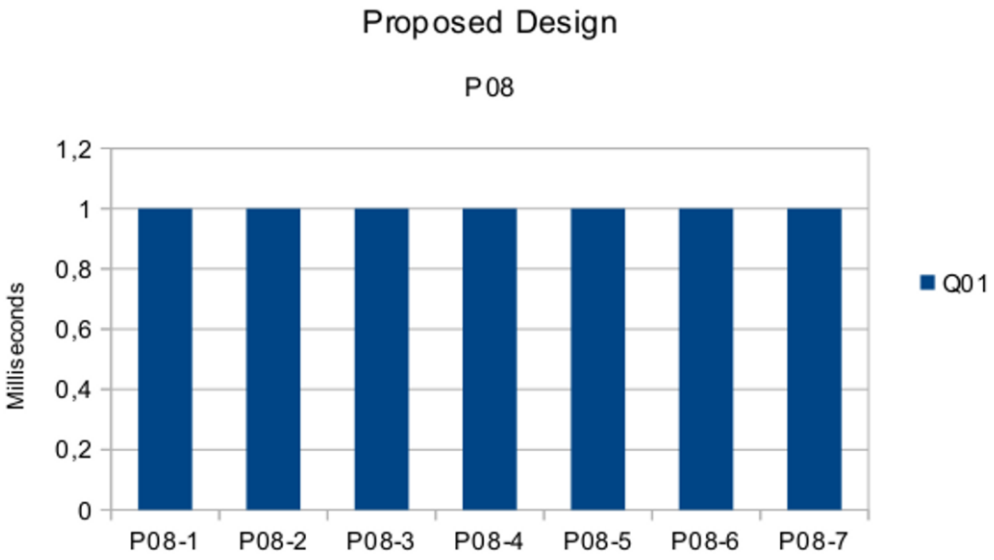


Ilustración 69: Resultados conjunto P08 - esquema propuesto

8.5.2.9 Conjunto P09

La Ilustración 70 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P09,* el cual corresponde a la ejecución de los conjuntos *I01* y *Q02* en paralelo.

Query	Q02	Returned samples
P09-1	1	1
P09-2	1	60
P09-3	1	3.600
P09-4	1	10.800
P09-5	1	21.600
P09-6	1	43.200
P09-7	1	86.400

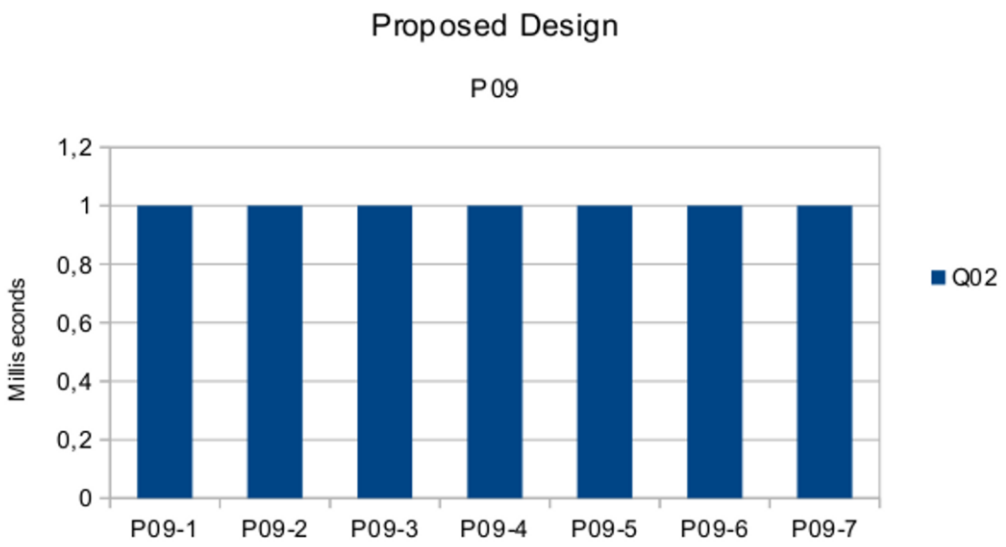


Ilustración 70: Resultados conjunto P09 - esquema propuesto

8.5.2.10 Conjunto P10

La Ilustración 71 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P10,* el cual corresponde a la ejecución de los conjuntos *I01* y *Q03* en paralelo.

Query	Q03	Returned samples
P10-1	6	1
P10-2	1	60
P10-3	1	3.600
P10-4	1	10.800
P10-5	1	21.600
P10-6	1	43.200
P10-7	1	86.400

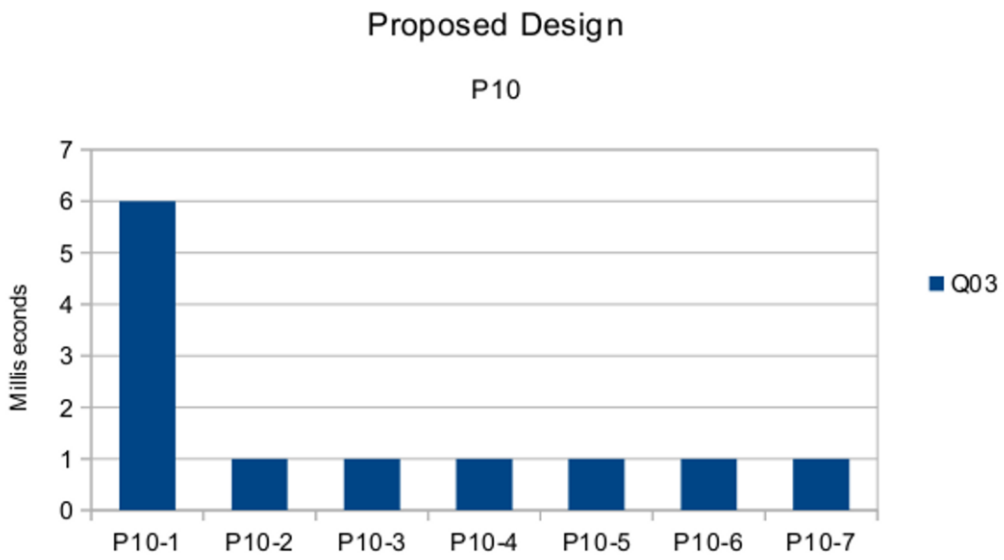


Ilustración 71: Resultados conjunto P10 - esquema propuesto

8.5.2.11 Conjunto P11

La Ilustración 72 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P11,* el cual corresponde a la ejecución de los conjuntos *I01*, *Q01 y Q02* en paralelo.

Query	Q01	Q02	Returned samples
P11-1	1	1	2
P11-2	1	1	120
P11-3	1	1	7.200
P11-4	4	1	21.600
P11-5	14	0	43.200
P11-6	1	1	86.400
P11-7	1	1	172.800

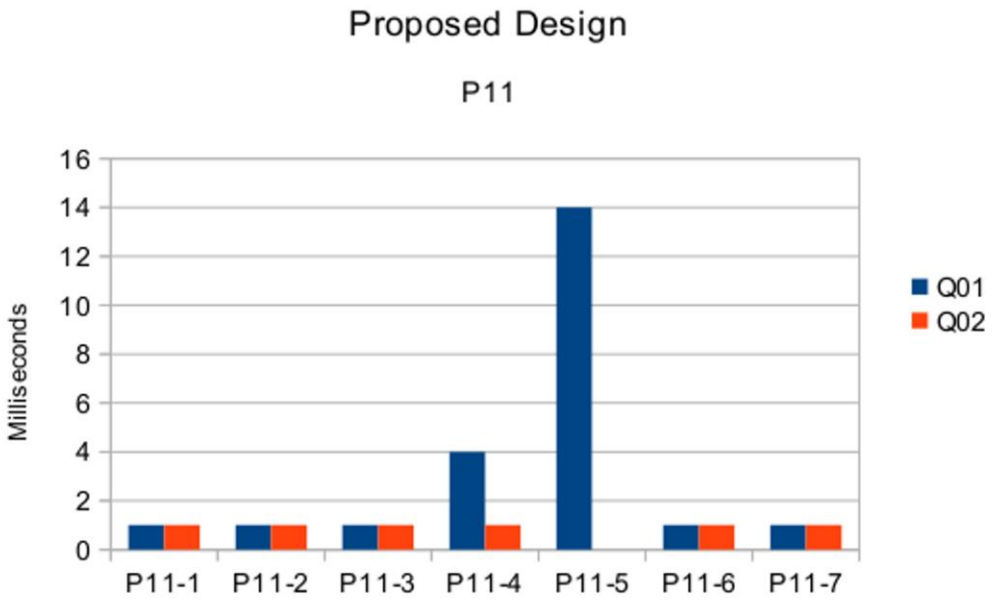


Ilustración 72: Resultados conjunto P11 - esquema propuesto

8.5.2.12 Conjunto P12

La Ilustración 73 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P12,* el cual corresponde a la ejecución de los conjuntos *I01*, *Q01* y *Q03* en paralelo.

Query	Q01	Q03	Returned samples
P12-1	1	1	2
P12-2	2	1	120
P12-3	1	1	7.200
P12-4	18	1	21.600
P12-5	1	0	43.200
P12-6	1	11	86.400
P12-7	1	1	172.800

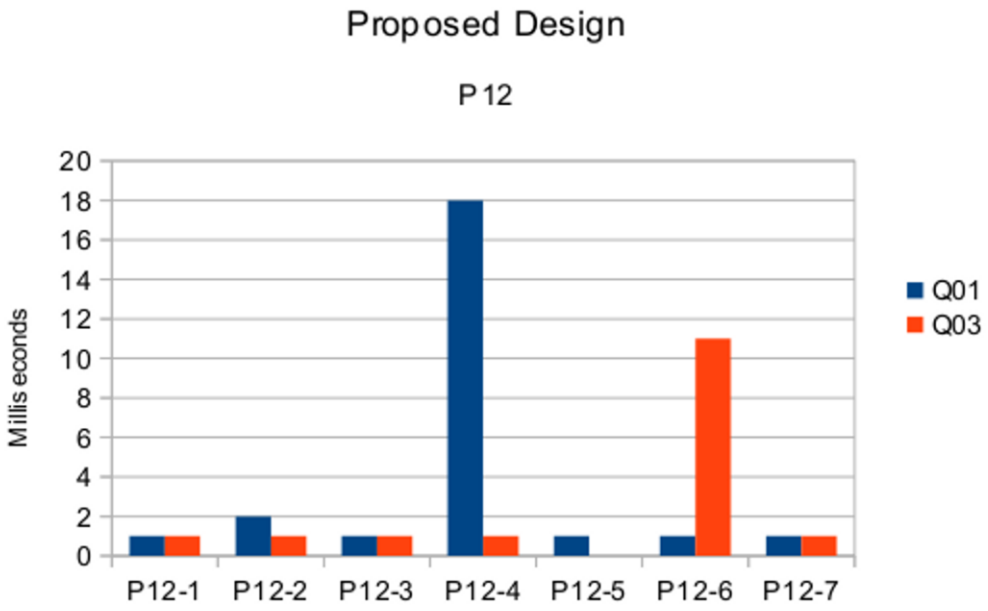


Ilustración 73: Resultados conjunto P12 - esquema propuesto

8.5.2.13 Conjunto P13

La Ilustración 74 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P13,* el cual corresponde a la ejecución de los conjuntos *I01*, *Q02* y *Q03* en paralelo.

Query	Q02	Q03	Returned samples
P13-1	1	1	2
P13-2	2	1	120
P13-3	1	1	7.200
P13-4	1	1	21.600
P13-5	1	1	43.200
P13-6	1	2	86.400
P13-7	7	2	172.800

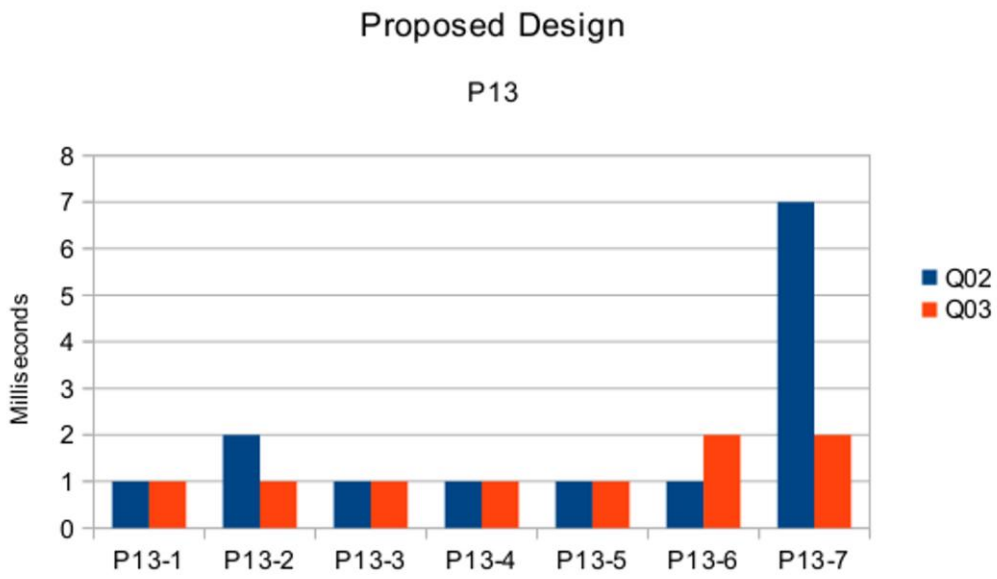


Ilustración 74: Resultados conjunto P13 - esquema propuesto

8.5.2.14 Conjunto P14

La Ilustración 75 presenta los resultados de la ejecución de cada una de las consultas del conjunto *P14,* el cual corresponde a la ejecución de los conjuntos *I01*, *Q01, Q02* y *Q03* en paralelo.

Query	Q01	Q02	Q03	Returned samples
P14-1	1	1	1	3
P14-2	1	1	1	180
P14-3	1	1	1	10.800
P14-4	1	1343	1175	32.400
P14-5	1	1	1	64.800
P14-6	1	1	1	129.600
P14-7	1	1	1	259.200

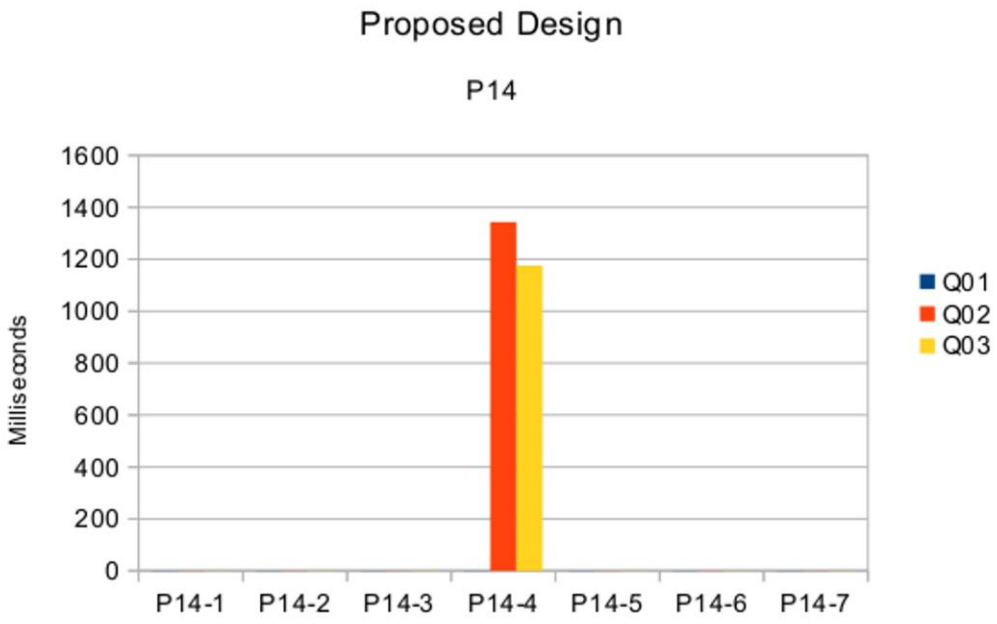


Ilustración 75: Resultados conjunto P14 - esquema propuesto

8.5.2.15 Análisis de rendimiento del conjunto Q01

La Ilustración 76 presenta una comparación en el rendimiento de todas las pruebas en las que está presente el conjunto de extracción *Q01*.

Query	P01	P04	P05	P07	P08	P11	P12	P14	Returned samples
Q01-1	28	1	1	12	1	1	1	1	1
Q01-2	22	1	1	1	1	1	2	1	60
Q01-3	0	0	1	1	1	1	1	1	3.600
Q01-4	1	1	1	1	1	4	18	1	10.800
Q01-5	1	1	3	1	1	14	1	1	21.600
Q01-6	1	1	0	1	1	1	1	1	43.200
Q01-7	1	1	1	0	1	1	1	1	86.400

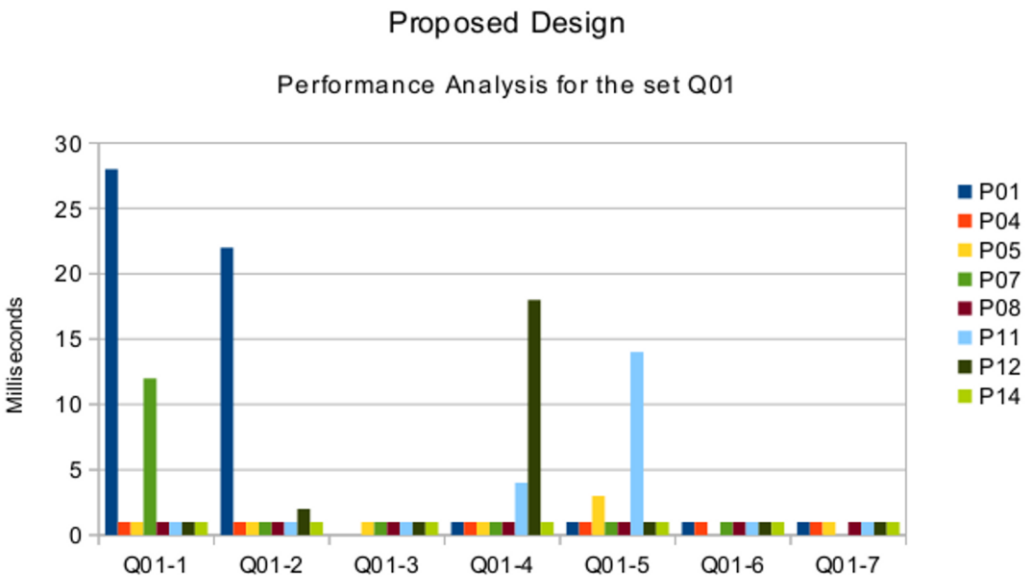


Ilustración 76: Análisis de rendimiento para el conjunto Q01 del esquema propuesto

8.5.2.16 Análisis de rendimiento del conjunto Q02

La Ilustración 77 presenta una comparación en el rendimiento de todas las pruebas en las que está presente el conjunto de extracción *Q02*.

Query	P02	P04	P06	P07	P09	P11	P13	P14	Returned samples
Q02-1	1	1	1	5	1	1	1	1	1
Q02-2	1	1	1	1	1	1	2	1	60
Q02-3	1	0	1	1	1	1	1	1	3.600
Q02-4	1	1	1	1	1	1	1	1343	10.800
Q02-5	1	1	1	1	1	0	1	1	21.600
Q02-6	1	1	0	1	1	1	1	1	43.200
Q02-7	1	0	1	1	1	1	7	1	86.400

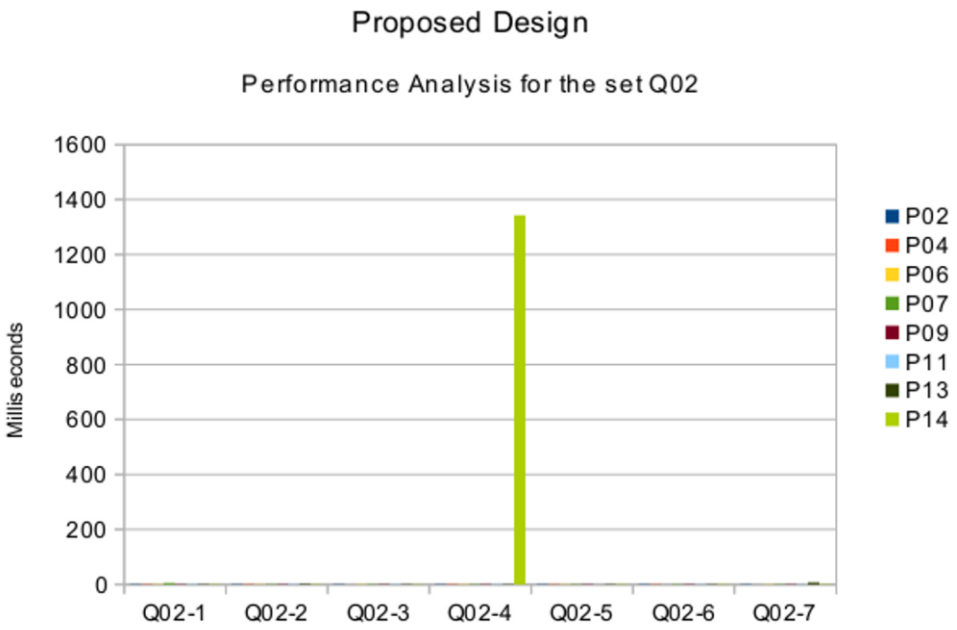


Ilustración 77: Análisis de rendimiento para el conjunto Q02 del esquema propuesto

El tiempo en la extracción de Q02-4 en la prueba P14 tomó más que al resto de sus pares. Esto ocurrió porque MongoDB quitó del caché los datos de Q02-4 y se dirigió al disco para obtenerlos, considerando, además, que la prueba P14 consiste de 1 flujo constante de inserciones (lo que se traduce en escrituras en disco) y 3 conjuntos de selección (lo que se traduce a lecturas de caché o disco) provocaron un rendimiento atípico.

8.5.2.17 Análisis de rendimiento del conjunto Q03

La Ilustración 78 presenta una comparación en el rendimiento de todas las pruebas en las que está presente el conjunto de extracción *Q03*.

Query	P03	P05	P06	P07	P10	P12	P13	P14	Returned samples
Q03-1	31	5	0	1	6	1	1	1	1
Q03-2	17	0	1	1	1	1	1	1	60
Q03-3	22	1	0	1	1	1	1	1	3.600
Q03-4	2	1	0	0	1	1	1	1175	10.800
Q03-5	22	1	1	1	1	0	1	1	21.600
Q03-6	19	0	0	1	1	11	2	1	43.200
Q03-7	1	1	1	1	1	1	2	1	86.400

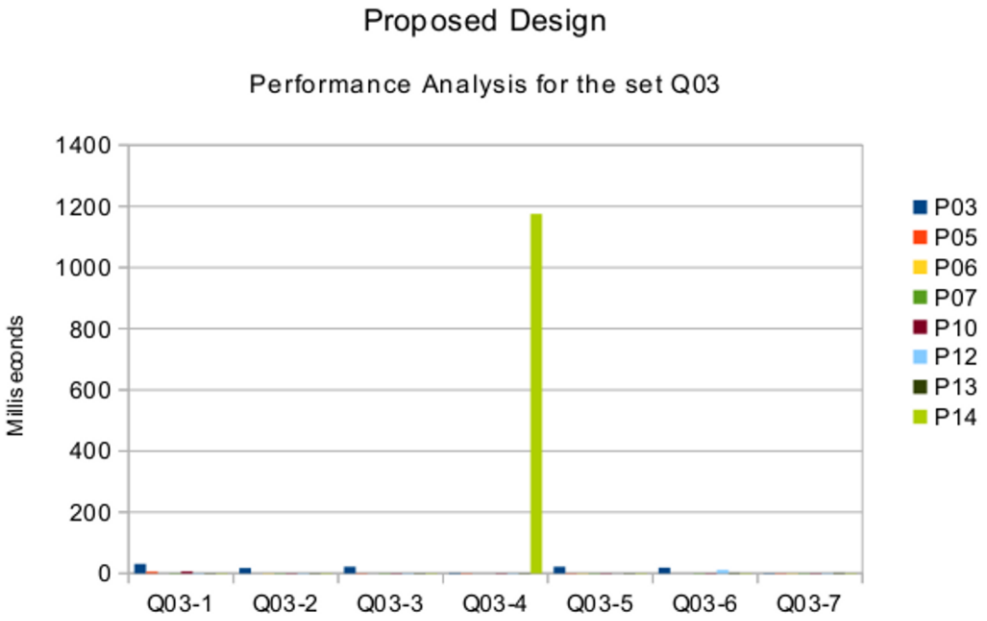


Ilustración 78: Análisis de rendimiento para el conjunto Q03 del esquema propuesto

Este caso es el mismo que en la sección anterior, *8.5.2.16 Análisis de rendimiento del conjunto Q02*. La Ilustración 78 muestra que Q03-4 de la prueba P14 tomó más tiempo del esperado, resultando en una muestra atípica.

8.6 Estadísticas

La Tabla 18 presenta un análisis estadístico del rendimiento obtenido por el conjunto de selección S[n]. La unidad de medida son segundos.

Medida Estadística	S01	S02	S03
Valor mínimo	1,44	1,11	0,3
Valor máximo	995,87	906,08	187,55
Promedio	120,92	140,91	13,24
Desviación estándar	229,46	236,27	37,68

Tabla 18: Estadísticas del conjunto S[n]

La Tabla 19 presenta un análisis estadístico del rendimiento obtenido por el conjunto de selección Q[n]. La unidad de medida son milisegundos.

Medida Estadística	Q01	Q02	Q03
Valor mínimo	0	0	0
Valor máximo	28	1343	1175
Promedio	2,63	25,09	24,11
Desviación estándar	5,43	179,32	156,72

Tabla 19: Estadísticas del conjunto Q[n]

La Tabla 20 presenta un análisis comparativo del rendimiento general obtenido por los conjuntos de selección S[n] y Q[n]. La unidad de medida son milisegundos.

Medida Estadística	Diseño Actual	Diseño Propuesto
Valor mínimo	300	0
Valor máximo	995.870	1.343
Promedio	91.690	17,27
Desviación estándar	198.380	137,10

Tabla 20: Comparación de medidas estadísticas de los diseños evaluados

Como se aprecia en los resultados estadísticos, el diseño propuesto tiene un rendimiento muy superior al diseño actual.

Por otro lado, el rendimiento de selección no fue afectado en los casos en que existía un flujo de escritura hacia la base de datos.

8.7 Desempeño del índice

En la sección *5.8.3 Uso de índices*, se cuantifico el rendimiento de los índices involucrados en las consultas de selección sobre el diseño actual. Ahora que ya se han realizado las pruebas sobre el diseño propuesto se procede a verificar el funcionamiento de su índice.

La Ilustración 79 muestra el desempeño obtenido al realizar una selección de un documento completo, es decir, con todos las muestras de un día. Como se aprecia en la tabla y según lo visto en la sección *4.2.6.3 El índice óptimo*, *n = nscanned = nscannedObjects*. Esto significa, que el número de objetos retornados es el mismo que los escaneados, por lo tanto para esta consulta el índice es óptimo.

```
mongos> db.monitorData_2.find({"metadata.antenna": "Antenna_70", "metadata.monitorPoint": "MP_1", "metadata.component": "Component_1", "metad
ata.date": "2013-2-10" }, { _id:1 }).explain();
{
        "clusteredType" : "ParallelSort",
        "n" : 1,
        "nChunkSkips" : 0,
        "nYields" : 0,
        "nscanned" : 1,
        "nscannedAllPlans" : 1,
        "nscannedObjects" : 1,
        "nscannedObjectsAllPlans" : 1,
```

Ilustración 79: Desempeño del índice en el diseño propuesto al extraer un documento

Continuando con el análisis, la Ilustración 80 presenta el rendimiento de una consulta de selección sobre un rango de documentos, que en este caso son todos los documentos de la colección que pertenezcan a la antena "Antenna_70", componente "Component_1" y punto de monitoreo "MP_1". La sección *8.4.1 Dominio de datos* del diseño propuesto dice que se insertaron 11 días de datos, por lo que en teoría, la consulta de selección debería retornar 11 documentos.

```
mongos> db.monitorData_2.find({"metadata.antenna": "Antenna_70", "metadata.monitorPoint": "MP_1", "metadata.component": "Component_1" }, { _i
d:1 }).explain();
{
        "clusteredType" : "ParallelSort",
        "n" : 11,
        "nChunkSkips" : 0,
        "nYields" : 0,
        "nscanned" : 11,
        "nscannedAllPlans" : 11,
        "nscannedObjects" : 11,
        "nscannedObjectsAllPlans" : 11,
```

Ilustración 80: Desempeño del índice en el diseño propuesto al extraer múltiples documentos

Como se aprecia en la Ilustración 80, el número de documentos encontrados es 11 y además *n* = *nscanned* = *nscannedObjects.* Por lo tanto, el índice es un índice óptimo.

Por otro lado, no sólo el rendimiento mejoró, sino que también el tamaño del índice disminuyó. Como se aprecia en la Ilustración 81, el tamaño del índice es de 193 MB, en contraste con los 1.851 GB del diseño actual.

```
mongos> db.monitorData_2.stats(1024*1024)
{
        "sharded" : true,
        "ns" : "OneDocumentPerformanceTest.monitorData_2",
        "count" : 1104950,
        "numExtents" : 705,
        "size" : 1364398,
        "storageSize" : 1387532,
        "totalIndexSize" : 193,
```

Ilustración 81: Tamaño del índice en el esquema propuesto

Pero el tamaño de los índices del diseño actual corresponde a 3 meses de datos y el diseño propuesto sólo a 11 días. Por lo que para poder compararlos es necesaria una proyección de índice.

Si se considera un crecimiento lineal del índice en el diseño propuesto en 9 veces, se tendrían 99 días de datos, lo cual sí es comparable a los 90 días del diseño actual.

Entonces,

Índice diseño propuesto = 193 *MB*9* = 1.737 *MB* = *1,6* GB

Índices diseño actual = 1,8 *TB* =1843,2 *GB*

Por lo tanto,

Disminución del tamaño = 1-(1,6 *GB*/ 1843,2 *GB*)= 0,999131944

El diseño propuesto utiliza un 99,91 % menos espacio físico en almacenar los índices.

8.8 Desempeño del sharding

La Ilustración 82 muestra el estado del balanceo de datos y como se aprecia, los nodos *mongodb1* y *mongodb2* prácticamente poseen la misma cantidad de chunks, 889 y 891 respectivamente.

```
mongos> use OneDocumentPerformanceTest
switched to db OneDocumentPerformanceTest
mongos> sh.status()
--- Sharding Status ---
  sharding version: { "_id" : 1, "version" : 3 }
        {  "_id" : "OneDocumentPerformanceTest",  "partitioned" : true,  "primary" : "mongodb2" }
                OneDocumentPerformanceTest.monitorData_2
                        shard key: { "metadata.antenna" : 1, "metadata.monitorPoint" : 1 }
                        chunks:
                                mongodb1        889
                                mongodb2        891
                        too many chunks to print, use verbose if you want to force print
        {  "_id" : "monitorData_2",  "partitioned" : true,  "primary" : "mongodb2" }

mongos>
```

Ilustración 82: Estado del sharding en el diseño propuesto

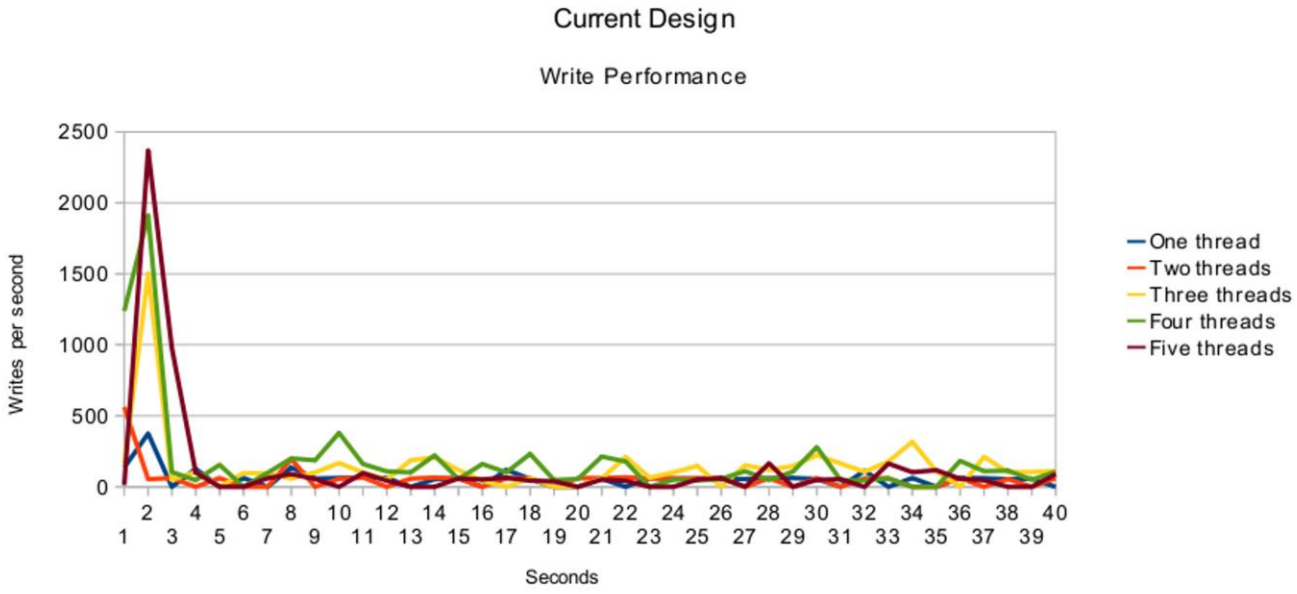
Por lo tanto, la llave-compartida *{ antenna, monitorPoint }* es óptima.

8.9 Inserciones

A continuación se presentan los gráficos sobre el desempeño en escritura alcanzado en MongoDB. Todas las pruebas utilizan entre uno y cinco hilos que realizan inserciones o actualizaciones paralelas.

8.9.1 Diseño Actual

La Ilustración 83 muestra el rendimiento obtenido en la inserción de documentos en el diseño actual.

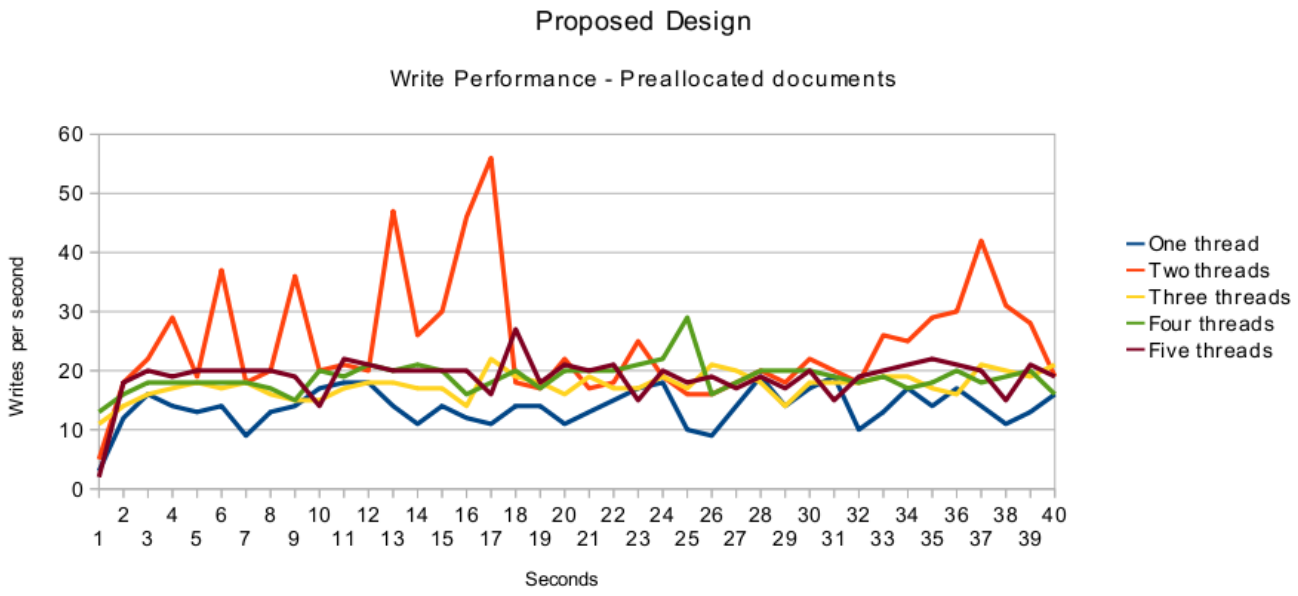


Stats	One thread	Two threads	Three threads	Four threads	Five threads
Mininimal Value	0,00	0,00	0,00	0,00	0,00
Maximal Value	375,00	562,00	1503,00	1911,00	2368,00
Average	56,90	54,75	142,93	183,78	127,48
Standar deviation	64,88	90,55	232,16	342,04	394,57
Total inserts	2276	2190	5717	7351	5099

Ilustración 83: Rendimiento en la inserción de muestras - Esquema Actual

8.9.2 Diseño propuesto

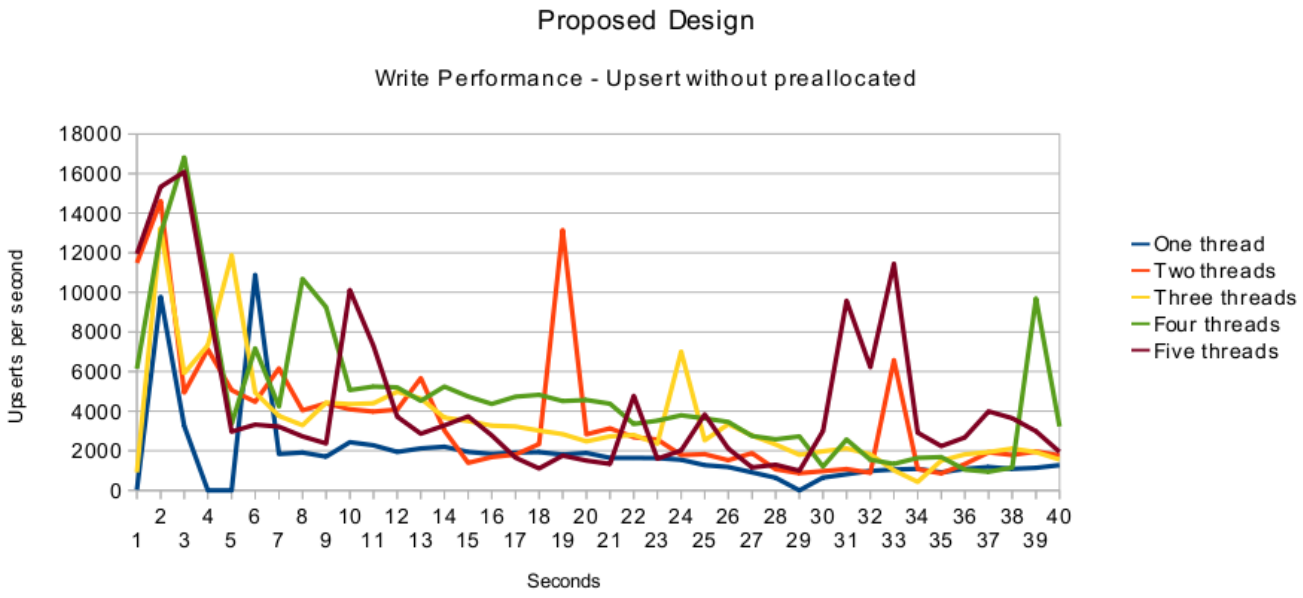
La Ilustración 84 muestra el rendimiento alcanzado al insertar documentos pre-localizados en MongoDB. Los documentos insertados tienen todos los segundos del día y un tamaño promedio de 1,3 MB.



Stats	One thread	Two threads	Three threads	Four threads	Five threads
Mininimal Value	3,00	5,00	11,00	13,00	2,00
Maximal Value	19,00	56,00	22,00	29,00	27,00
Average	13,80	24,60	17,53	18,83	18,90
Standar deviation	3,21	9,96	2,20	2,49	3,59
Total inserts	552	984	701	753	756

Ilustración 84: Rendimiento en la inserción de documentos pre-localizados - Esquema Propuesto

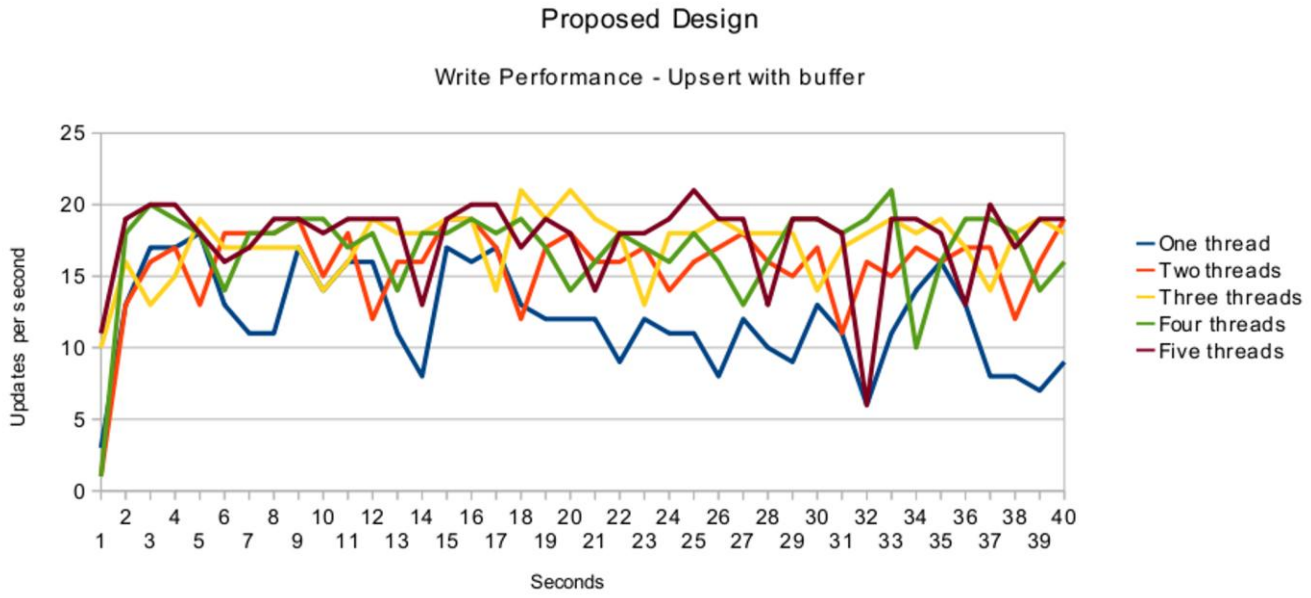
La Ilustración 85 muestra el rendimiento al insertar muestras a un documento que no ha sido pre-localizado. Como se aprecia en el gráfico el rendimiento es inestable, esto ocurre porque una vez que se han insertado varias muestras a un documento y el espacio físico en el que se encuentra se llena, MongoDB debe re-ubicarlo en otro sector con mayor memoria contigua, mientras se realiza este proceso de traslación la base de datos no insertará valores en el documento, haciendo que el desempeño disminuya considerablemente.



Stats	One thread	Two threads	Three threads	Four threads	Five threads
Mininimal Value	0,00	847,00	429,00	933,00	989,00
Maximal Value	10.875,00	14.609,00	13.246,00	16.814,00	16.079,00
Average	1.829,88	3.593,85	3.543,50	4.752,63	4.422,30
Standar deviation	2.099,67	3.238,27	2.597,12	3.429,22	3.903,90
Total upserts	73.195,00	143.754,00	141.740,00	190.105,00	176.892,00

Ilustración 85: Rendimiento en la inserción de muestras sin pre-localizar los documentos - Esquema Propuesto

La Ilustración 86 muestra el rendimiento al actualizar los atributos de un documento pre-localizado, consultando a la base de datos si el documento existe o no. Cuando una muestra es consumida desde ActiveMQ por el TMC, éste debe saber si el documento en el cual la muestra será insertada existe o no. Si el documento no existe, entonces el TMC debe insertar un documento pre-localizado con valores estándares e incluir a la muestra en cuestión. Por otro lado, si el documento existe, el TMC sólo debe actualizar el atributo correspondiente con el valor de la muestra. Este proceso de consultar a la base de datos por un documento antes de ejecutar una inserción o actualización tiene una penalidad en el rendimiento y el gráfico lo muestra claramente, el mejor desempeño es de tan solo 17,68 actualizaciones por segundo.



Stats	One thread	Two threads	Three threads	Four threads	Five threads
Mininimal Value	3,00	1,00	10,00	1,00	6,00
Maximal Value	18,00	19,00	21,00	21,00	21,00
Average	12,05	15,75	17,23	16,85	17,68
Standar deviation	3,52	3,16	2,29	3,36	2,92
Total updates	482,00	630,00	689,00	674,00	707,00

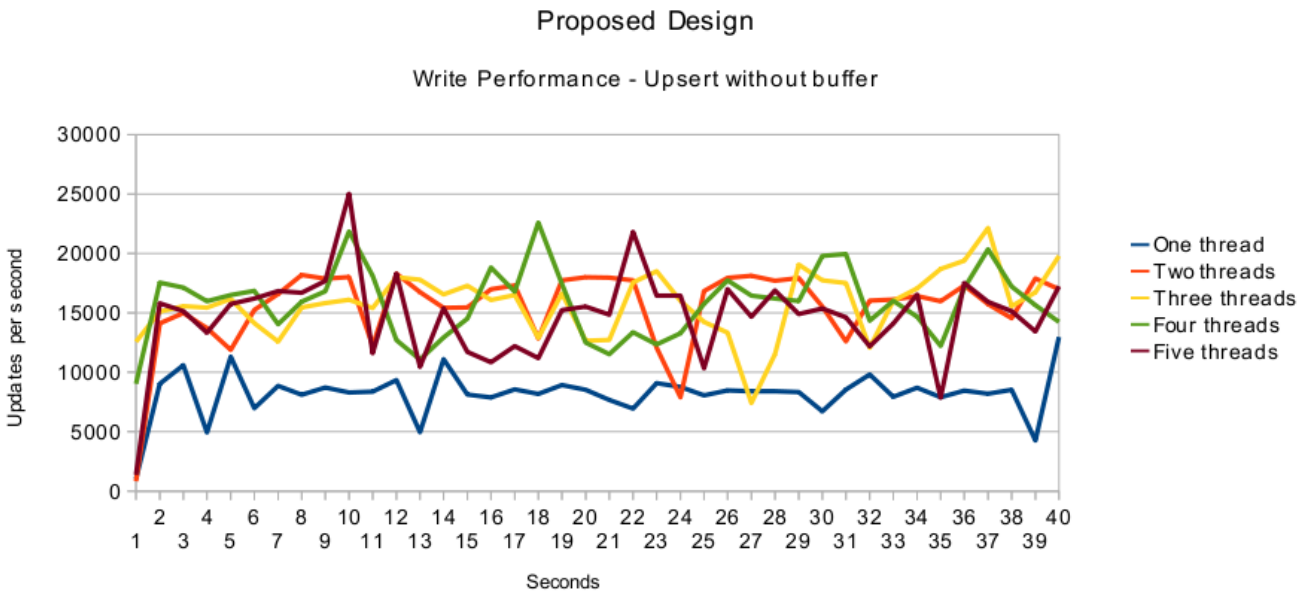
Ilustración 86: Rendimiento en la actualización de muestras utilizando verificación de documentos

La Ilustración 87 muestra el rendimiento al actualizar los atributos sin consultar a la base de datos si el documento donde se actualizará existe o no. Esta prueba es la misma que la presentada en la Ilustración 86, pero con la verificación de documentos desactivada.

El rendimiento alcanzado en esta prueba es muy superior a todos los demás, esto se debe por dos razones:

1. No se consulta a la base de datos por el documento a actualizar.
2. El documento en el cual se actualizará está pre-localizado.

Para lograr el rendimiento alcanzado se deben asegurar ambos puntos, es decir, asegurar que el documento en el que se almacenará la muestra se encuentre pre-localizado para no tener que consultar por él ni que MongoDB deba moverlo para buscar más espacio físico.



Stats	One thread	Two threads	Three threads	Four threads	Five threads
Mininimal Value	1.025,00	823,00	7.408,00	9.029,00	1.360,00
Maximal Value	12.970,00	18.288,00	22.136,00	22.600,00	25.005,00
Average	8.197,15	15.542,68	15.791,83	15.829,70	14.737,18
Standar deviation	1.951,15	3.305,47	2.700,04	2.922,59	3.765,49
Total updates	327.886,00	621.707,00	631.673,00	633.188,00	589.487,00

Ilustración 87: Rendimiento en la actualización de muestras sin verificación de documentos

8.10 Conclusión

Según los análisis realizados a los resultados de las pruebas, el rendimiento de las consultas de selección del diseño propuesto son considerablemente superiores a las del diseño actual.

También se aprecia que los 1,3 MB de tamaño de los documentos pre-ubicados no afectan al rendimiento de MongoDB en la extracción de datos, lo cual responde a la interrogante planteada en la sección *6.3 Seleccionando el diseño*.

Las pruebas demuestran que los índices son los óptimos y que se están escaneando la misma cantidad de documentos solicitados.

En cuanto al sharding, también se está ejecutando correctamente y la llave seleccionada demuestra ser óptima en la distribución de datos a través de los nodos del cluster.

Por otro lado, las pruebas demuestran que el enfoque de agregar una muestra a un documento pre-localizado es el único que cumple con el requerimiento de las ocho mil inserciones por segundo, demostrando ser la mejor opción.

Por lo tanto, se concluye que el diseño propuesto si cumple con los requerimientos del TMC.

9 FACTIBILIDAD

9.1 Introducción

Este capítulo de factibilidad tiene por objetivo determinar los recursos necesarios para implementar la solución descrita en este estudio, tanto técnicos como económicos.

9.2 Factibilidad Técnica

El objetivo de este estudio es determinar si técnicamente MongoDB es capaz de cumplir con los requerimientos del TMC, lo cual quedó demostrado en el capítulo .

En cuanto a infraestructura, ALMA ya posee un cluster de MongoDB y también cuenta con servidores extra para añadir más capacidad si fuese necesario.

Por otro lado, el departamento de computación de ALMA también cuenta con recurso humano calificado para implantar la solución propuesta en este estudio.

9.3 Factibilidad Económica

A continuación se incluye un análisis de costo asociados a la implementación de la solución propuesta.

9.3.1 Horas hombre

La Tabla 21 presenta la estimación del costo de un ingeniero que implementaría el proyecto. Se han considerado dos categorías, basados en la experiencia del ingeniero, Junior (sin experiencia o poca) y Senior (3-5 años de experiencia). A un ingeniero junior le tomaría 3 meses implementar el proyecto, en cambio a un senior sólo 1 mes.

Ingeniero	Valor mensual	duración	Total
Junior	$ 700.000	3 meses	$ 2.100.000
Senior	$ 1.400.000	1 meses	$ 1.400.000

Tabla 21: Estimación de costo del recurso humano

9.3.2 Licencias de software

MongoDB es software libre, lo cual significa que no hay que pagar por su uso. El sistema operativo instalado sobre el hardware de los nodos de MongoDB es CentOS 5.8, el cual es software libre, por lo que tampoco hay que pagar por su uso. Por otro lado, el servidor utilizado para las pruebas funciona con Red Hat 6 Enterprise Edition, pero ALMA cuenta con la versión corporativa, la cual tiene un valor independiente de la cantidad de máquinas en las que se utilice, por lo tanto la licencia del sistema operativo se considera como un costo hundido.

9.4 Conclusión factibilidad

Como ya se ha visto, técnicamente, el proyecto es factible de implementar y en términos económicos, también, ya que los costos son bajos y no suponen una cuantiosa inversión, principalmente gracias al software libre y sus beneficios que hacen que la implementación de este proyecto no requiera de la compra de licencias de software.

10 CONCLUSIÓN

La cantidad de opciones NoSQL que el mercado ofrece es bastante, de las más variadas y para todas las necesidades. Dentro de ellas se encuentra una en particular, MongoDB, que es considerada un líder de este movimiento tecnológico, principalmente, por las capacidades de rendimiento que ofrece, el estado avanzado de su desarrollo, la completa y amplia documentación oficial y el soporte de su comunidad, hacen de este motor de datos una atractiva solución. Si a lo anterior se le añade que MongoDB es OpenSource, lo cual significa cero costo de licencias, su atractivo es aún mayor.

El objetivo de ADC al realizar este estudio es determinar si MongoDB es capaz de obtener el rendimiento que el TMC requiere para almacenar los datos monitoreados de las antenas, lo cual quedo demostrado que sí.

Este estudio ha cubierto todos los aspectos planteados en los objetivos. Primero, se analizó la arquitectura actual de manejo de los datos de monitoreo, estudiando todos los aspectos relacionados a ella, tales como: la estructura de los mensajes, los clob, las muestras, entre otros. Luego, se analizó el estado del diseño actual del almacenamiento de los datos en MongoDB, obteniendo datos duros sobre la situación de los índices y el sharding.

Por otro lado, el capítulo de diseño planteó tres enfoques para resolver el problema de rendimiento del diseño actual, dentro de los cuales se analizaron sus ventajas y desventajas, y realizando estimaciones para cada uno de ellos. Finalmente, fueron esas estimaciones las que sentaron las bases para seleccionar el enfoque más apropiado para cumplir con los requerimientos del TMC.

El capítulo de pruebas fue crucial para determinar si realmente el diseño propuesto eras capaz de cumplir el rendimiento requerido. Los resultados obtenidos son satisfactorios y se han presentado en gráficos, además se han obtenido medidas estadísticas de ellos. Finalmente, fueron estos resultados los que determinaron que el diseño *"Un documento diario por punto de monitoreo"* si cumple con los requisitos de rendimiento que el TMC necesita y su desempeño es superior al del diseño actualmente implementado, demostrando que es posible extraer todas las muestras diarias de una variable en unos pocos milisegundos.

Durante el desarrollo de este trabajo se ha podido comprobar dos aspectos fundamentales a la hora de implementar una solución. El primero, la importancia de analizar y comprender el problema al que se enfrenta y considerar todas las variables involucradas en el. El segundo, y no menos importante, conocer las características y funcionalidades de la tecnología que se implantará, ya que de esta forma es posible ajustar el problema enfrentado a las herramientas que se poseen. Estos dos

aspectos son los que determinaran, en gran medida, el éxito o fracaso de un proyecto.

Un aspecto fundamental que dio un giro a este proyecto fue la adopción de un cambio en la estructura de almacenamiento de los datos. La forma en que el diseño *"Un documento diario por punto de monitoreo"* almacena los datos no es algo muy natural si se proviene de un paradigma relacional, sin embargo fue una decisión acertada e impulsó a los buenos resultados del estudio.

Pequeñas modificaciones pueden hacer grandes y notorios cambios, sobre todo cuando se trata de grandes volúmenes de información, no sólo en el almacenamiento, sino también en su procesamiento. La adopción de tecnologías NoSQL está comenzando y abren un sin fin de posibilidades por explotar.

11 BIBLIOGRAFÍA

[1]. **Tiwari, Shashank.** *Professional NoSQL.* Canada : John Wiley & Sons, Inc., 2011. ISBN: 978-0-470-94224-6.

[2]. **Wikipedia.** Wikipedia. *Wikipedia.* [En línea] [Citado el: 27 de Febrero de 2013.] http://es.wikipedia.org/wiki/Web_2.0.

[3]. **Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach.** *Bigtable: A Distributed Storage System for Structured Data.* s.l. : Google, Inc, 2006.

[4]. *The Google File System.* **Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.** Boston : Google, inc, 19 de Octubre de 2003.

[5]. **Burrows, Mike.** *he Chubby lock service for loosely-coupled distributed systems.* s.l. : Google Inc. http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//archive/chubby-osdi06.pdf.

[6]. **Ghemawat, Jeffrey Dean and Sanjay.** *MapReduce: Simplied Data Processing on Large Clusters.* s.l. : Google, Inc, 2004.

[7]. **Requena, Cristian.** NoSql. [En línea] 1 de Abril de 2010. [Citado el: 28 de Febrero de 2013.] (http://www.nosql.es/blog/nosql/que-es-nosql.html).

[8]. **10gen.** MongoDB. [En línea] 2013. [Citado el: 15 de Enero de 2013.] http://docs.mongodb.org/manual/.

[9]. **Davias, Jesse Jiryu.** Empysquare. [En línea] 2012 de Octubre de 8. http://emptysquare.net/blog/optimizing-mongodb-compound-indexes/.

[10]. **10gen.** MongoDB. [En línea] 2013. [Citado el: 7 de Febrero de 2013.] http://docs.mongodb.org/manual/applications/indexes/#create-indexes-to-support-your-queries.

[11]. **Chodorow, Kristina.** kchodorow. [En línea] 10gen, 4 de Enero de 2011. [Citado el: 2 de Febrero de 2013.]http://www.kchodorow.com/blog/2011/01/04/how-to-choose-a-shard-key-the-card-game/.

[12]. **10gen.** MongoDB. [En línea] 10gen, 2013. [Citado el: 16 de Diciembre de 2012.] http://docs.mongodb.org/manual/faq/storage/.

[13]. **Oracle.** Oracle. [En línea] 2003. [Citado el: 3 de Noviembre de

2012.] http://docs.oracle.com/javaee/1.4/api/javax/jms/MapMessage.html.

[14]. **10gen.** MongoDB. [En línea] 10gen, 2013. [Citado el: 15 de Febrero de 2013.] http://docs.mongodb.org/manual/use-cases/pre-aggregated-reports/) .

12 APENDICE I - SCRIPT CONSULTA

Aplicación escrita en Python para consultar datos en un rango de fechas al esquema actual. También entrega el número de registros obtenidos y el tiempo de respuesta.

Este código no fue escrito por el autor de este estudio y fue facilitado por los administradores de sistemas de ALMA.

```
#!/usr/bin/python


import pymongo
import argparse
import datetime
import json
import time
import random


def myargs():
    now = datetime.datetime.now()
    defserv = "mongo-r1.osf.alma.cl"
    rnd = random.randint(0,1000)
    parser = argparse.ArgumentParser(description='I do mongo queries.')
    parser.add_argument('--server', metavar='server',
                        default = defserv, help='server address')
    parser.add_argument('--componentname', metavar='componentname',
                        help='name of the component')
    parser.add_argument('--devicename', metavar='devicename',
                        help='name of the device')
    parser.add_argument('--monitorpoint', metavar='monitorpoint',
```

```
                help='name of the monitorpoint')
    parser.add_argument('--datestart', metavar='datestart',
                default = now, help='date in yyyy-mm-dd HH:mm:ss')
    parser.add_argument('--dateend', metavar='dateend',
                default = now, help='date in yyyy-mm-dd HH:mm:ss')
    parser.add_argument('--outfile', default = "outfile%s.json" % (rnd), help='filename')
    parser.add_argument('--path', default = ".", help='path for the file with no final /')
    parser.add_argument('--mail', default = None, help='mail to the user')
    return parser

def str2date(adata):
    part = adata.split(" ")
    fechas = part[0].split("-")
    horas = part[1].split(":")
  #print horas
  #print fechas
    return datetime.datetime(int(fechas[0]), int(fechas[1]), int(fechas[2]), int(horas[0]), int(horas[1]), int(horas[2]))

def somemail(astr):
    part = astr.split("@")
    return part[0]

def go(mail, filename, path, comp, dev, mpname, start, end, acoll="", adb="MONDB", srvr="mongo-r1.osf.alma.cl", output="array"):
    startime = datetime.datetime.now()
    connection = pymongo.Connection(srvr, 27017)
```

```
    db = connection[adb]
    collection = db["monitorPoints"]
    if (dev == None):
        query = {"date": { '$gte': str2date(start), '$lte': str2date(end)}, "componentName" : comp}
        dev='All_Devices'
    else:
        query = {"date": { '$gte': str2date(start), '$lte': str2date(end)}, "componentName" : comp, "monitorPointName": mpname , "propertyName" : dev }
    print query
  #print query
    datas=collection.find(query)
  #dthandler = lambda obj: obj.isoformat() if isinstance(obj, datetime.datetime) else None
    out = []
    for data in datas:
    #print data
    #out.append([data['Date'].strftime("%d/%m/%y %H:%M:%S"), data['meanStat']])
      #out.append([data['date'].strftime("%d/%m/%y %H:%M:%S"),data['acsTime'], data['componentName'],data['propertyName'],data['monitorPointName'],data['monitorValue']])
      out.append([data['date'].isoformat(),data['acsTime'], data['componentName'],data['propertyName'],data['monitorPointName'],data['monitorValue']])
    f = open( "%s/%s" % (path, filename), "w")
    for line in out:
      f.write("%s\n" % (line))
    f.close()
    connection.close()
    endtime = datetime.datetime.now()
    extra = ""
```

```
    if mail != None:
      user = somemail(mail)
      extra = "Dear %s,\n" %(user.capitalize())
      filename = "http://apo-01.dhcp.alma.cl:8000/show/?data=%s" % (filename)
    msg = "%s%s %s elements from %s in %s in %s" % (extra, datas.count(), dev, comp, endtime-startime, filename)
    print msg


def main():
    parse = myargs()
    options = parse.parse_args()
    go(options.mail, options.outfile, options.path, options.componentname, options.devicename, options.monitorpoint, options.datestart, options.dateend)


if __name__ == '__main__':
   main()
```

13 APENDICE II - SCRIPT CONJUNTO S01

```
#!/bin/bash
#
# Este script contiene y ejecuta las consultas del conjunto S01, utilizadas
# para el probar el rendimiento de las base de datos
# con el esquema actual
#
# Author: Leonel Peña

MONGO_SCRIPT="./query.py"
PATH_LOGS="/home/kilua/UBB/Tesis/doc/MongoDB/test/logs/QueriesTest1/"
LOG_NAME=$PATH_LOGS$(date +%Y-%m-%d)".txt"
LOG_GENERAL=$PATH_LOGS"result-"$(date +%Y-%m-%d)".txt"
QUERY="$MONGO_SCRIPT --datestart \"%s\" --dateend \"%s\" --component %s --devicename %s --monitorpoint %s"
QUERY2="$MONGO_SCRIPT --datestart \"%s\" --dateend \"%s\" --component %s --devicename %s --monitorpoint %s --path $PATH_LOGS --outfile %s"
COMMANDS=( )
QUERY_NAME=( )

Component='CONTROL/DV10/LLC'
Device='P_DET'
MonitorPoint='P_DET'

if [ -e $MONGO_SCRIPT ]; then
    if [ ! -x $MONGO_SCRIPT ]; then
```

```
        echo "The script $MONGO_SCRIPT does not have execute permissions"
        exit 1
    fi
else
    echo "The script $MONGO_SCRIPT was not found"
    exit 1
fi


function makeAQuery {
    # We build the query
    if [ $# -eq 6 ]; then
#        echo -n $(printf "$QUERY" "$1" "$2" "$3" "$4" "$5")
        echo -n $(printf "$QUERY2" "$1" "$2" "$3" "$4" "$5" "$6")
#    elif [ $# -eq 4 ]; then
#        echo -n $(printf "$QUERY" "$1" "$2" "$3" "$4")
    else
        printf "Use: sh QueryTest.sh [DATE START] [DATE END] [COMPONENT NAME] [DEVICE NAME] [MONITOR POINT] [OUTPUT FILE]\n"
        printf "Example: sh QueryTest.sh \"2012-11-10 00:01:00\" \"2012-11-10 23:59:00\" CONTROL/DV10/LLC P_DET P_DET test.json\n"
        exit 1
    fi
}
# 1) Para diez minutos de datos
DateStart1='2012-11-14 22:34:00'
DateEnd1='2012-11-14 22:44:00'
```

```
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 1\"" )

COMMAND_1=$(makeAQuery "$DateStart1" "$DateEnd1" "$Component" "$Device" "$MonitorPoint" "Query_1.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_1" )


# 2) Para veinte minutos de datos (NO SE UTILIZA)

#DateStart2='2012-10-29 15:05:00'

#DateEnd2='2012-10-29 15:25:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 2\"" )

COMMAND_2=$(makeAQuery "$DateStart2" "$DateEnd2" "$Component" "$Device" "$MonitorPoint" "Query_2.json")

#COMMANDS=( "${COMMANDS[@]}" "$COMMAND_2" )

COMMANDS=( "${COMMANDS[@]}" "The query2 has not been set" )


# 3) Para treinta minutos de datos

DateStart3='2012-10-12 23:40:00'

DateEnd3='2012-10-13 00:10:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 3\"" )

COMMAND_3=$(makeAQuery "$DateStart3" "$DateEnd3" "$Component" "$Device" "$MonitorPoint" "Query_3.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_3" )


# 4) Para sesenta minutos de datos

DateStart4='2012-09-10 16:15:00'

DateEnd4='2012-09-10 17:15:00'
```

```
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 4\"" )

COMMAND_4=$(makeAQuery "$DateStart4" "$DateEnd4" "$Component" "$Device" "$MonitorPoint" "Query_4.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_4" )


# 5) Para tres horas de datos

DateStart5='2012-11-10 00:30:00'

DateEnd5='2012-11-10 03:30:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 5\"" )

COMMAND_5=$(makeAQuery "$DateStart5" "$DateEnd5" "$Component" "$Device" "$MonitorPoint" "Query_5.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_5" )


# 6) Para seis horas de datos

DateStart6='2012-11-20 14:01:00'

DateEnd6='2012-11-20 20:01:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 6\"" )

COMMAND_6=$(makeAQuery "$DateStart6" "$DateEnd6" "$Component" "$Device" "$MonitorPoint" "Query_6.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_6" )


# 7) Para doce horas de datos

DateStart7='2012-10-05 10:30:00'

DateEnd7='2012-10-05 22:30:00'
```

```
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 7\"" )
COMMAND_7=$(makeAQuery "$DateStart7" "$DateEnd7" "$Component" "$Device" "$MonitorPoint" "Query_7.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_7" )

# 8) Para veinticuatro horas de datos
DateStart8='2012-09-25 08:00:00'
DateEnd8='2012-09-26 08:00:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 8\"" )
COMMAND_8=$(makeAQuery "$DateStart8" "$DateEnd8" "$Component" "$Device" "$MonitorPoint" "Query_8.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_8" )

######################
# ***************** #
# ****  MAIN  ***** #
# ***************** #
######################

if [ $# -eq 1 ]; then

    echo $(date --rfc-3339=seconds)": Starting MongoDB Test"
    echo $(date --rfc-3339=seconds)": Starting MongoDB Test" 1>>$LOG_GENERAL

    if [ $1 == "--all" ]; then
```

```
        # Executing all queries
        len=${#COMMANDS[*]}

        for ((i=0; i<len; i++))
        do
            eval ${QUERY_NAME[$i]}  1>>$LOG_GENERAL
            eval ${COMMANDS[$i]} 1>>$LOG_GENERAL
#             echo ${COMMANDS[$i]}
        done
    else
        # Executing one query
        eval ${QUERY_NAME[($1-1)]} 1>>$LOG_GENERAL
        eval ${COMMANDS[($1-1)]} 1>>$LOG_GENERAL
#         echo ${COMMANDS[($1-1)]}
    fi
    echo $(date --rfc-3339=seconds)": The test has been executed"
    echo $(date --rfc-3339=seconds)": The test has been executed" 1>>$LOG_GENERAL
else
    printf "Usage: $0 [QUERY NUMBER] | --all\n
            Examples:\n
            For execute the query one\n
                $0 1\n
            For execute all queries\n
                $0 --all\n\n"
fi
```

14 APENDICE III - SCRIPT CONJUNTO S02

```
#!/bin/bash
#
# Este script contiene y ejecuta las consultas del conjunto S02, utilizadas
# para el probar el rendimiento de las base de datos
# con el esquema actual
# Author: Leonel Peña

MONGO_SCRIPT="./query.py"
PATH_LOGS="/home/kilua/UBB/Tesis/doc/MongoDB/test/logs/QueriesTest2/"
LOG_NAME=$PATH_LOGS$(date +%Y-%m-%d)".txt"
LOG_GENERAL=$PATH_LOGS"result-"$(date +%Y-%m-%d)".txt"
QUERY="$MONGO_SCRIPT --datestart \"%s\" --dateend \"%s\" --component %s --devicename %s --monitorpoint %s"
QUERY2="$MONGO_SCRIPT --datestart \"%s\" --dateend \"%s\" --component %s --devicename %s --monitorpoint %s --path $PATH_LOGS --outfile %s"
COMMANDS=( )
QUERY_NAME=( )

Component='CONTROL/DV06/LLC'
Device='POL_MON4'
MonitorPoint='POL_MON4'

if [ -e $MONGO_SCRIPT ]; then
    if [ ! -x $MONGO_SCRIPT ]; then
        echo "The script $MONGO_SCRIPT does not have execute permissions"
```

```
        exit 1
    fi
else
    echo "The script $MONGO_SCRIPT was not found"
    exit 1
fi

function makeAQuery {
    # We build the query
    if [ $# -eq 6 ]; then
#        echo -n $(printf "$QUERY" "$1" "$2" "$3" "$4" "$5")
        echo -n $(printf "$QUERY2" "$1" "$2" "$3" "$4" "$5" "$6")
    else
        printf "Use: sh QueryTest.sh [DATE START] [DATE END] [COMPONENT NAME] [DEVICE NAME] [MONITOR POINT] [OUTPUT FILE]\n"
        printf "Example: sh QueryTest.sh \"2012-11-10 00:01:00\" \"2012-11-10 23:59:00\" CONTROL/DV10/LLC P_DET P_DET test.json\n"
        exit 1
    fi
}

# 1) Para diez minutos de datos
DateStart1='2012-10-14 12:20:00'
DateEnd1='2012-10-14 12:30:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 1\"" )
COMMAND_1=$(makeAQuery "$DateStart1" "$DateEnd1" "$Component" "$Device" "$MonitorPoint"
```

```
"Query_1.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_1" )

# 2) Para veinte minutos de datos (NO SE UTILIZA)
#DateStart2='2012-10-29 15:05:00'
#DateEnd2='2012-10-29 15:25:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 2\"" )
COMMAND_2=$(makeAQuery "$DateStart2" "$DateEnd2" "$Component" "$Device" "$MonitorPoint"
"Query_2.json")
#COMMANDS=( "${COMMANDS[@]}" "$COMMAND_2" )
COMMANDS=( "${COMMANDS[@]}" "The query2 has not been set" )

# 3) Para treinta minutos de datos
DateStart3='2012-09-25 05:40:00'
DateEnd3='2012-09-25 06:10:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 3\"" )
COMMAND_3=$(makeAQuery "$DateStart3" "$DateEnd3" "$Component" "$Device" "$MonitorPoint"
"Query_3.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_3" )

# 4) Para sesenta minutos de datos
DateStart4='2012-11-10 19:13:00'
DateEnd4='2012-11-10 20:13:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 4\"" )
```

```
COMMAND_4=$(makeAQuery "$DateStart4" "$DateEnd4" "$Component" "$Device" "$MonitorPoint" "Query_4.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_4" )


# 5) Para tres horas de datos

DateStart5='2012-11-19 23:30:00'

DateEnd5='2012-11-20 02:30:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 5\"" )

COMMAND_5=$(makeAQuery "$DateStart5" "$DateEnd5" "$Component" "$Device" "$MonitorPoint" "Query_5.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_5" )


# 6) Para seis horas de datos

DateStart6='2012-10-03 14:01:00'

DateEnd6='2012-10-03 20:01:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 6\"" )

COMMAND_6=$(makeAQuery "$DateStart6" "$DateEnd6" "$Component" "$Device" "$MonitorPoint" "Query_6.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_6" )


# 7) Para doce horas de datos

DateStart7='2012-09-29 06:05:00'

DateEnd7='2012-09-29 18:05:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 7\"" )
```

```
COMMAND_7=$(makeAQuery "$DateStart7" "$DateEnd7" "$Component" "$Device" "$MonitorPoint" "Query_7.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_7" )

# 8) Para veinticuatro horas de datos
DateStart8='2012-09-29 01:00:00'
DateEnd8='2012-09-30 01:00:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 8\"" )
COMMAND_8=$(makeAQuery "$DateStart8" "$DateEnd8" "$Component" "$Device" "$MonitorPoint" "Query_8.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_8" )

######################
# ***************** #
# ****  MAIN  ***** #
# ***************** #
######################

if [ $# -eq 1 ]; then

   echo $(date --rfc-3339=seconds)": Starting MongoDB Test"
   echo $(date --rfc-3339=seconds)": Starting MongoDB Test" 1>>$LOG_GENERAL

   if [ $1 == "--all" ]; then
      # Executing all queries
      len=${#COMMANDS[*]}
```

```
        for ((i=0; i<len; i++))
        do
            eval ${QUERY_NAME[$i]}  1>>$LOG_GENERAL
            eval ${COMMANDS[$i]} 1>>$LOG_GENERAL
#             echo ${COMMANDS[$i]}
        done

    else
        # Executing one query
        eval ${QUERY_NAME[($1-1)]} 1>>$LOG_GENERAL
        eval ${COMMANDS[($1-1)]} 1>>$LOG_GENERAL
#         echo ${COMMANDS[($1-1)]}
    fi

    echo $(date --rfc-3339=seconds)": The test has been executed"
    echo $(date --rfc-3339=seconds)": The test has been executed" 1>>$LOG_GENERAL
else
    printf "Usage: $0 [QUERY NUMBER] | --all\n
            Examples:\n
            For execute the query one\n
                $0 1\n
            For execute all queries\n
                $0 --all\n\n"
fi
```

15 APENDICE IV - SCRIPT CONJUNTO S03

```
#!/bin/bash
#
# Este script contiene y ejecuta las consultas del conjunto S03, utilizadas
# para el probar el rendimiento de las base de datos
# con el esquema actual
#
# Author: Leonel Peña

MONGO_SCRIPT="./query.py"
PATH_LOGS="/home/kilua/UBB/Tesis/doc/MongoDB/test/logs/QueriesTest3/"
LOG_NAME=$PATH_LOGS$(date +%Y-%m-%d)".txt"
LOG_GENERAL=$PATH_LOGS"result-"$(date +%Y-%m-%d)".txt"
QUERY="$MONGO_SCRIPT --datestart \"%s\" --dateend \"%s\" --component %s --devicename %s --monitorpoint %s"
QUERY2="$MONGO_SCRIPT --datestart \"%s\" --dateend \"%s\" --component %s --devicename %s --monitorpoint %s --path $PATH_LOGS --outfile %s"
COMMANDS=( )
QUERY_NAME=( )
Component='CONTROL/DV10/FrontEnd/Cryostat'
Device='GATE_VALVE_STATE'
MonitorPoint='GATE_VALVE_STATE'

if [ -e $MONGO_SCRIPT ]; then
    if [ ! -x $MONGO_SCRIPT ]; then
        echo "The script $MONGO_SCRIPT does not have execute permissions"
```

```
        exit 1
    fi
else
    echo "The script $MONGO_SCRIPT was not found"
    exit 1
fi

function makeAQuery {
    # We build the query
    if [ $# -eq 6 ]; then
#        echo -n $(printf "$QUERY" "$1" "$2" "$3" "$4" "$5")
        echo -n $(printf "$QUERY2" "$1" "$2" "$3" "$4" "$5" "$6")
    else
        printf "Use: sh QueryTest.sh [DATE START] [DATE END] [COMPONENT NAME] [DEVICE NAME] [MONITOR POINT] [OUTPUT FILE]\n"
        printf "Example: sh QueryTest.sh \"2012-11-10 00:01:00\" \"2012-11-10 23:59:00\" CONTROL/DV10/LLC P_DET P_DET test.json\n"
        exit 1
    fi
}

# 1) Para diez minutos de datos
DateStart1='2012-09-18 09:20:00'
DateEnd1='2012-09-18 09:30:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 1\"" )
COMMAND_1=$(makeAQuery "$DateStart1" "$DateEnd1" "$Component" "$Device" "$MonitorPoint"
```

```
"Query_1.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_1" )

# 2) Para veinte minutos de datos (NO SE UTILIZA)
#DateStart2='2012-10-29 15:05:00'
#DateEnd2='2012-10-29 15:25:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 2\"" )
COMMAND_2=$(makeAQuery "$DateStart2" "$DateEnd2" "$Component" "$Device" "$MonitorPoint"
"Query_2.json")
#COMMANDS=( "${COMMANDS[@]}" "$COMMAND_2" )
COMMANDS=( "${COMMANDS[@]}" "The query2 has not been set" )

# 3) Para treinta minutos de datos
DateStart3='2012-10-20 13:25:00'
DateEnd3='2012-10-20 13:55:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 3\"" )
COMMAND_3=$(makeAQuery "$DateStart3" "$DateEnd3" "$Component" "$Device" "$MonitorPoint"
"Query_3.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_3" )

# 4) Para sesenta minutos de datos
DateStart4='2012-10-05 11:00:00'
DateEnd4='2012-10-05 12:00:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 4\"" )
```

```
COMMAND_4=$(makeAQuery "$DateStart4" "$DateEnd4" "$Component" "$Device" "$MonitorPoint" "Query_4.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_4" )


# 5) Para tres horas de datos

DateStart5='2012-11-21 11:09:00'

DateEnd5='2012-11-21 14:09:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 5\"" )

COMMAND_5=$(makeAQuery "$DateStart5" "$DateEnd5" "$Component" "$Device" "$MonitorPoint" "Query_5.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_5" )


# 6) Para seis horas de datos

DateStart6='2012-09-25 23:45:00'

DateEnd6='2012-09-26 05:45:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 6\"" )

COMMAND_6=$(makeAQuery "$DateStart6" "$DateEnd6" "$Component" "$Device" "$MonitorPoint" "Query_6.json")

COMMANDS=( "${COMMANDS[@]}" "$COMMAND_6" )


# 7) Para doce horas de datos

DateStart7='2012-11-16 15:43:00'

DateEnd7='2012-11-17 03:43:00'


QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 7\"" )
```

```
COMMAND_7=$(makeAQuery "$DateStart7" "$DateEnd7" "$Component" "$Device" "$MonitorPoint" "Query_7.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_7" )

# 8) Para veinticuatro horas de datos
DateStart8='2012-09-09 01:50:00'
DateEnd8='2012-09-10 01:50:00'

QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 8\"" )
COMMAND_8=$(makeAQuery "$DateStart8" "$DateEnd8" "$Component" "$Device" "$MonitorPoint" "Query_8.json")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_8" )

######################
# ***************** #
# ****  MAIN  ***** #
# ***************** #
######################

if [ $# -eq 1 ]; then

   echo $(date --rfc-3339=seconds)": Starting MongoDB Test"
   echo $(date --rfc-3339=seconds)": Starting MongoDB Test" 1>>$LOG_GENERAL

   if [ $1 == "--all" ]; then
      # Executing all queries
      len=${#COMMANDS[*]}
```

```
        for ((i=0; i<len; i++))
        do
            eval ${QUERY_NAME[$i]}  1>>$LOG_GENERAL
            eval ${COMMANDS[$i]} 1>>$LOG_GENERAL
#             echo ${COMMANDS[$i]}
        done

    else
        # Executing one query
        eval ${QUERY_NAME[($1-1)]} 1>>$LOG_GENERAL
        eval ${COMMANDS[($1-1)]} 1>>$LOG_GENERAL
#         echo ${COMMANDS[($1-1)]}
    fi

    echo $(date --rfc-3339=seconds)": The test has been executed"
    echo $(date --rfc-3339=seconds)": The test has been executed" 1>>$LOG_GENERAL
else
    printf "Usage: $0 [QUERY NUMBER] | --all\n
            Examples:\n
            For execute the query one\n
                $0 1\n
            For execute all queries\n
                $0 --all\n\n"
fi
```

16 APENDICE V - SCRIPT CONJUNTO Q01

```
#!/bin/bash
#
# Este script contiene y ejecuta las consultas que se utilizan
# para el probar el rendimiento de las base de datos
# con el esquema propuesto
#
# Author: Leonel Peña
#
ANTENNA="Antenna_1"
COMPONENT='Component_1'
MONITOR_POINT='MP_1'

#HOST="localhost"
HOST="mongo-r1.osf.alma.cl"
DATABASE="OneDocumentPerformanceTest"
COLLECTION="monitorData_2"
RESULT_DIR="./result/QueriesTest1"

#QUERY="mongo --host $HOST $DATABASE --eval \"printjson(db.$COLLECTION.findOne({'metadata.antenna':'$ANTENNA','metadata.monitorPoint':'$MONITOR_POINT','metadata.component':'$COMPONENT','metadata.date':'%s'} %s))\" > %s"
QUERY="mongo --host $HOST $DATABASE --eval \"printjson(db.$COLLECTION.find({'metadata.antenna':'$ANTENNA','metadata.monitorPoint':'$MONITOR_POINT','metadata.component':'$COMPONENT','metadata.date':'%s'} %s).explain())\" > %s"
```

```
# Arguments:
#   $1  :   Date to consult
#   $2  :   Data projection. find() method format
#   $3  :   File to write the result
function makeAQuery {
    # We build the query
    if [ $# -eq 3 ]; then
        echo -n $(printf "$QUERY" "$1" "$2" "$RESULT_DIR/$3")
    else
        printf "Error: Illegal arguments number in makeAQuery function\n"
        exit 1
    fi
}
#
# 1) Granularidad de búsqueda: 1 segundo
#
Date="2013-2-2"
Projection=", {_id:1, 'hourly.19.32.15':1}"
File="1s.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 1\"" )
COMMAND_1=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_1" )
#
#
# 2) DESACTIVADA
```

```
#
# 3) Granularidad de búsqueda: 1 minuto
#
Date="2013-2-9"
Projection=", {_id:1, 'hourly.22.4':1}"
File="1m.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 3\"" )
COMMAND_3=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_3" )
#
# 4) Para sesenta minutos de datos del mismo component y variable que 1)
#
Date="2013-2-3"
Projection=", {_id:1, 'hourly.16':1}"
File="1h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 4\"" )
COMMAND_4=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_4" )
#
#
# 5) Para tres horas de datos de un componente y una sola variable
#
Date="2013-2-5"
Projection=", {_id:1, 'hourly.19':1, 'hourly.20':1, 'hourly.21':1}"
```

```
File="3h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 5\"" )
COMMAND_5=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_5" )
#
# 6) Para seis horas de datos del mismo component y variable que 1)
#
Date="2013-2-5"
Projection=", {_id:1, 'hourly.1':1, 'hourly.2':1, 'hourly.3':1, 'hourly.4':1, 'hourly.5':1, 'hourly.6':1}"
File="6h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 6\"" )
COMMAND_6=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_6" )
#
# 7) Para doce horas de datos del mismo component y variable que 1)
#
Date="2013-2-7"
Projection=", {_id:1, 'hourly.11':1, 'hourly.12':1, 'hourly.13':1, 'hourly.14':1, 'hourly.15':1, 'hourly.16':1,
'hourly.17':1, 'hourly.18':1, 'hourly.19':1, 'hourly.20':1, 'hourly.21':1, 'hourly.22':1}"
File="12h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 7\"" )
COMMAND_7=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_7" )
```

```
#
# 8) Para veinticuatro horas de datos del mismo component y variable que 1)
#
Date="2013-2-1"
Projection=""
File="24h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 8\"" )
COMMAND_8=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_8" )
#
#
######################
# ***************** #
# ****  MAIN  ***** #
# ***************** #
######################
#
if [ $# -eq 1 ]; then

    echo $(date --rfc-3339=seconds)": Starting MongoDB Test"
#    echo $(date --rfc-3339=seconds)": Starting MongoDB Test" 1>>$LOG_GENERAL

    if [ $1 == "--all" ]; then
        # Executing all queries
        len=${#COMMANDS[*]}
```

```
        for ((i=0; i<len; i++))
        do
#           eval ${QUERY_NAME[$i]}  1>>$LOG_GENERAL
#           eval ${COMMANDS[$i]} 1>>$LOG_GENERAL
            eval ${COMMANDS[$i]}
#           echo ${COMMANDS[$i]}
        done

    else
        # Executing one query
        eval ${QUERY_NAME[($1-1)]} 1>>$LOG_GENERAL
        eval ${COMMANDS[($1-1)]} 1>>$LOG_GENERAL
#       echo ${COMMANDS[($1-1)]}
    fi

    echo $(date --rfc-3339=seconds)": The test has been executed"
#    echo $(date --rfc-3339=seconds)": The test has been executed" 1>>$LOG_GENERAL
else
    printf "Usage: $0 [QUERY NUMBER] | --all\n
            Examples:\n
            For execute the query one\n
               $0 1\n
            For execute all queries\n
               $0 --all\n\n"
fi
```

17 APENDICE VI - SCRIPT CONJUNTO Q02

```
#!/bin/bash
#
# Este script contiene y ejecuta las consultas que se utilizan
# para el probar el rendimiento de las base de datos
# con el esquema propuesto
#
# Author: Leonel Peña
#
ANTENNA="Antenna_6"
COMPONENT='Component_23'
MONITOR_POINT='MP_19'

#HOST="localhost"
HOST="mongo-r1.osf.alma.cl"
DATABASE="OneDocumentPerformanceTest"
COLLECTION="monitorData_2"
RESULT_DIR="./result/QueriesTest2"

#QUERY="mongo --host $HOST $DATABASE --eval \"printjson(db.$COLLECTION.findOne({'metadata.antenna':'$ANTENNA','metadata.monitorPoint':'$MONITOR_POINT','metadata.component':'$COMPONENT','metadata.date':'%s'} %s))\" > %s"
QUERY="mongo --host $HOST $DATABASE --eval \"printjson(db.$COLLECTION.find({'metadata.antenna':'$ANTENNA','metadata.monitorPoint':'$MONITOR_POINT','metadata.component':'$COMPONENT','metadata.date':'%s'} %s).explain())\" > %s"
```

```
# Arguments:
#   $1  :   Date to consult
#   $2  :   Data projection. find() method format
#   $3  :   File to write the result
function makeAQuery {
    # We build the query
    if [ $# -eq 3 ]; then
        echo -n $(printf "$QUERY" "$1" "$2" "$RESULT_DIR/$3")
    else
        printf "Error: Illegal arguments number in makeAQuery function\n"
        exit 1
    fi
}
#
# 1) Granularidad de búsqueda: 1 segundo
#
Date="2013-2-8"
Projection=", {_id:1, 'hourly.15.58.59':1}"
File="1s.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 1\"" )
COMMAND_1=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_1" )
#
#
# 2) DESACTIVADA
```

```
#
# 3) Granularidad de búsqueda: 1 minuto
#
Date="2013-2-7"
Projection=", {_id:1, 'hourly.4.0':1}"
File="1m.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 3\"" )
COMMAND_3=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_3" )
#
# 4) Para sesenta minutos de datos del mismo component y variable que 1)
#
Date="2013-2-1"
Projection=", {_id:1, 'hourly.1':1}"
File="1h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 4\"" )
COMMAND_4=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_4" )
#
#
# 5) Para tres horas de datos de un componente y una sola variable
#
Date="2013-2-9"
Projection=", {_id:1, 'hourly.5':1, 'hourly.6':1, 'hourly.7':1}"
```

```
File="3h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 5\"" )
COMMAND_5=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_5" )
#
# 6) Para seis horas de datos del mismo component y variable que 1)
#
Date="2013-2-3"
Projection=", {_id:1, 'hourly.11':1, 'hourly.12':1, 'hourly.13':1, 'hourly.14':1, 'hourly.15':1, 'hourly.16':1}"
File="6h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 6\"" )
COMMAND_6=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_6" )
#
# 7) Para doce horas de datos del mismo component y variable que 1)
#
Date="2013-2-8"
Projection=", {_id:1, 'hourly.3':1, 'hourly.4':1, 'hourly.5':1, 'hourly.6':1, 'hourly.7':1, 'hourly.8':1,
'hourly.9':1, 'hourly.10':1, 'hourly.11':1, 'hourly.12':1, 'hourly.13':1, 'hourly.14':1}"
File="12h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 7\"" )
COMMAND_7=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_7" )
```

```
#
# 8) Para veinticuatro horas de datos del mismo component y variable que 1)
#
Date="2013-2-10"
Projection=""
File="24h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 8\"" )
COMMAND_8=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_8" )
#
#
######################
# ***************** #
# ****  MAIN  ***** #
# ***************** #
######################
#
if [ $# -eq 1 ]; then

    echo $(date --rfc-3339=seconds)": Starting MongoDB Test"
#    echo $(date --rfc-3339=seconds)": Starting MongoDB Test" 1>>$LOG_GENERAL

    if [ $1 == "--all" ]; then
        # Executing all queries
        len=${#COMMANDS[*]}
```

```
        for ((i=0; i<len; i++))
        do
#           eval ${QUERY_NAME[$i]}  1>>$LOG_GENERAL
#           eval ${COMMANDS[$i]} 1>>$LOG_GENERAL
           eval ${COMMANDS[$i]}
#           echo ${COMMANDS[$i]}
        done

    else
        # Executing one query
        eval ${QUERY_NAME[($1-1)]} 1>>$LOG_GENERAL
        eval ${COMMANDS[($1-1)]} 1>>$LOG_GENERAL
#       echo ${COMMANDS[($1-1)]}
    fi

    echo $(date --rfc-3339=seconds)": The test has been executed"
#    echo $(date --rfc-3339=seconds)": The test has been executed" 1>>$LOG_GENERAL
else
    printf "Usage: $0 [QUERY NUMBER] | --all\n
            Examples:\n
            For execute the query one\n
               $0 1\n
            For execute all queries\n
               $0 --all\n\n"
fi
```

18 APENDICE VII - SCRIPT CONJUNTO Q03

```
#!/bin/bash
#
# Este script contiene y ejecuta las consultas que se utilizan
# para el probar el rendimiento de las base de datos
# con el esquema propuesto
#
# Author: Leonel Peña
#
ANTENNA="Antenna_5"
COMPONENT='Component_40'
MONITOR_POINT='MP_34'

#HOST="localhost"
HOST="mongo-r2.osf.alma.cl"
DATABASE="OneDocumentPerformanceTest"
COLLECTION="monitorData_2"
RESULT_DIR="./result/QueriesTest3"

#QUERY="mongo --host $HOST $DATABASE --eval \"printjson(db.$COLLECTION.findOne({'metadata.antenna':'$ANTENNA','metadata.monitorPoint':'$MONITOR_POINT','metadata.component':'$COMPONENT','metadata.date':'%s'} %s))\" > %s"
QUERY="mongo --host $HOST $DATABASE --eval \"printjson(db.$COLLECTION.find({'metadata.antenna':'$ANTENNA','metadata.monitorPoint':'$MONITOR_POINT','metadata.component':'$COMPONENT','metadata.date':'%s'} %s).explain())\" > %s"
```

```
# Arguments:
#   $1  :  Date to consult
#   $2  :  Data projection. find() method format
#   $3  :  File to write the result
function makeAQuery {
    # We build the query
    if [ $# -eq 3 ]; then
        echo -n $(printf "$QUERY" "$1" "$2" "$RESULT_DIR/$3")
    else
        printf "Error: Illegal arguments number in makeAQuery function\n"
        exit 1
    fi
}
#
# 1) Granularidad de búsqueda: 1 segundo
#
Date="2013-2-6"
Projection=", {_id:1, 'hourly.13.28.9':1}"
File="1s.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 1\"" )
COMMAND_1=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_1" )
#
#
# 2) DESACTIVADA
```

```
#
# 3) Granularidad de búsqueda: 1 minuto
#
Date="2013-2-5"
Projection=", {_id:1, 'hourly.8.11':1}"
File="1m.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 3\"" )
COMMAND_3=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_3" )
#
# 4) Para sesenta minutos de datos del mismo component y variable que 1)
#
Date="2013-2-9"
Projection=", {_id:1, 'hourly.23':1}"
File="1h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 4\"" )
COMMAND_4=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_4" )
#
#
# 5) Para tres horas de datos de un componente y una sola variable
#
Date="2013-2-2"
Projection=", {_id:1, 'hourly.11':1, 'hourly.12':1, 'hourly.13':1}"
```

```
File="3h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 5\"" )
COMMAND_5=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_5" )
#
# 6) Para seis horas de datos del mismo component y variable que 1)
#
Date="2013-2-10"
Projection=", {_id:1, 'hourly.3':1, 'hourly.4':1, 'hourly.5':1, 'hourly.6':1, 'hourly.7':1, 'hourly.8':1}"
File="6h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 6\"" )
COMMAND_6=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_6" )
#
# 7) Para doce horas de datos del mismo component y variable que 1)
#
Date="2013-2-1"
Projection=", {_id:1, 'hourly.4':1, 'hourly.5':1, 'hourly.6':1, 'hourly.7':1, 'hourly.8':1, 'hourly.9':1, 'hourly.10':1, 'hourly.11':1, 'hourly.12':1, 'hourly.13':1, 'hourly.14':1, 'hourly.15':1}"
File="12h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 7\"" )
COMMAND_7=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_7" )
```

```
#
# 8) Para veinticuatro horas de datos del mismo component y variable que 1)
#
Date="2013-2-6"
Projection=""
File="24h.txt"
#
QUERY_NAME=( "${QUERY_NAME[@]}" "echo \"Query 8\"" )
COMMAND_8=$(makeAQuery "$Date" "$Projection" "$File")
COMMANDS=( "${COMMANDS[@]}" "$COMMAND_8" )
#
#
######################
# ***************** #
# ****  MAIN  ***** #
# ***************** #
######################
#
if [ $# -eq 1 ]; then

    echo $(date --rfc-3339=seconds)": Starting MongoDB Test"
#    echo $(date --rfc-3339=seconds)": Starting MongoDB Test" 1>>$LOG_GENERAL

    if [ $1 == "--all" ]; then
        # Executing all queries
        len=${#COMMANDS[*]}
```

```
        for ((i=0; i<len; i++))
        do
#           eval ${QUERY_NAME[$i]}  1>>$LOG_GENERAL
#           eval ${COMMANDS[$i]} 1>>$LOG_GENERAL
            eval ${COMMANDS[$i]}
#           echo ${COMMANDS[$i]}
        done

    else
        # Executing one query
        eval ${QUERY_NAME[($1-1)]} 1>>$LOG_GENERAL
        eval ${COMMANDS[($1-1)]} 1>>$LOG_GENERAL
#       echo ${COMMANDS[($1-1)]}
    fi

    echo $(date --rfc-3339=seconds)": The test has been executed"
#    echo $(date --rfc-3339=seconds)": The test has been executed" 1>>$LOG_GENERAL
else
    printf "Usage: $0 [QUERY NUMBER] | --all\n
            Examples:\n
            For execute the query one\n
               $0 1\n
            For execute all queries\n
               $0 --all\n\n"
fi
```

19 APENDICE VIII - POSTPROCESAMIENTO DE DATOS

```
var flattenMapFunction = function() {
    for(var hour in this.hourly) {
        for(minute in this.hourly[hour]) {
            for(second in this.hourly[hour][minute]) {
                if( this.hourly[hour][minute][second].toString().indexOf("na")==-1 ) {
                    emit(this._id, this.hourly[hour][minute][second]);
                }
            }
        }
    }
};
var statsReduceFunction = function(idDocument, monitorPointValues) {
    reducedValue = {
        maxValue: Math.max.apply(Math, monitorPointValues),
        minValue: Math.min.apply(Math, monitorPointValues),
        average: 0,
        standarDeviation: 0
    };
    // Average
    reducedValue.average = (Array.sum(monitorPointValues))/monitorPointValues.length;
    // Standar deviation
    tmpValue = 0;
    for(var i=0; i<monitorPointValues.length; i++) {
        tmpValue += Math.pow((monitorPointValues[i]-reducedValue.average),2);
```

```
    }
    reducedValue.standarDeviation = Math.sqrt(tmpValue/(monitorPointValues.length-1));

    return reducedValue;
};
# Ejecución de las estadísticas
db.monitorData_[MONTH].mapReduce(
    flattenMapFunction,
    statsReduceFunction,
    { out: "monitorData_[MONTH].stats" }
)
```