NoSQL Database in the Cloud: Couchbase Server 2.0 on AWS

July 2013

Kyle Lichtenberg and Miles Ward

(Please consult **http://aws.amazon.com/whitepapers/** for the latest version of this whitepaper.)

Table of Contents

Abstract

Amazon Web Services (AWS) is a flexible, cost-effective, easy-to-use cloud computing platform. It gives you a variety of options to run your NoSQL workloads in the cloud. You can run your NoSQL workloads on Amazon DynamoDB, which is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. It also gives you an option to run your own NoSQL Database system on Amazon Elastic Compute Cloud (Amazon EC2). Couchbase is one of the most popular NoSQL database software that can run on Amazon EC2 and can be ideal if your application requires the unique benefits of a document oriented NoSQL system.

In this white paper, we provide an overview of general best practices for running Couchbase Server 2.0 on Amazon EC2 and examine important Couchbase implementation characteristics such as performance, durability, and security. We pay particular attention to identifying features of Amazon EC2 that can help support the scalability, high-availability, and fault tolerance of a Couchbase Server implementation.

Overview

Couchbase Server 2.0 is an easily scalable NoSQL database system that can provide consistent high-performance access to information, even during database upgrades and system-level administration tasks. Couchbase Server is a nonhierarchical, shared-nothing architecture with a single server type. Because of this flat design, you can easily resize your Couchbase Server system by adding and removing instances, even while the system is running and servicing requests.

Basic tips for Using Couchbase on Amazon EC2

- Couchbase Server includes a built-in object-level cache. By provisioning sufficient RAM for the working set (actively accessed data) of your application, you can consistently provide low latency with high throughput.
- Couchbase recommends allocating 60% of available memory on Couchbase instances for the working-set cache.
- We strongly recommend using 64-bit Amazon EC2 instances for production use, as they provide additional RAM availability on each instance and therefore across the entire cluster.
- Couchbase uses an eventual persistence model: writes are asynchronous, and mutations are de-duplicated prior to storage. If additional I/O performance is required, moving from a standard EBS volume, to a Provisioned IOPS volume and Provisioned IOPS EBS-based RAID, using EBS-optimized instances and SSD-based instances, and adding additional Couchbase instances will help increase system performance.
- Couchbase Server supports multiple Couchbase data buckets on a system that will compartmentalize data into containers. Use separate buckets to support different replica configurations and separate SASL authentication rules for different data sets.
- You should ensure that you have enough physical memory to accommodate the Couchbase Server image and your data; otherwise, excessive disk I/O will significantly degrade performance.
- For production usage, you should create your own bucket structure to use instead of the default structure that Couchbase Server creates during setup and installation.
- Use uniform Amazon EC2 instance types in your cluster.

- Use Amazon EC2 Cluster Compute instances for enhanced cross-system communication and performance.
- Use Cross-Datacenter Replication (XDCR) for in-region replication to secondary Couchbase clusters in different availability zones to help create a highly available infrastructure. Couchbase clusters should not normally be distributed across regional boundaries.

Getting Started

The recommended and often easiest way to get started is to use the Couchbase Server AMIs, which are available within the AWS Marketplace.

> **Note**: In this guide, command line examples that begin with `$` are run in a standard Linux shell.

1. Visit the AWS Marketplace. Search for and then select the Couchbase Server AMI you want to use. When prompted, sign in to your AWS account.
2. Select the AWS Region where you want to launch your Couchbase Server instance in the Region section.
3. Choose your preferred instance type in the Amazon EC2 Instance Type section.
4. Choose a Key Pair to associate with the instance in the Key Pair section.
5. Select the key pair you want to use when connecting to the instance over Secure Shell (SSH) in the Key Pair section.
6. Click **Launch with 1-Click**. This will create your new instance.
7. After the instance has been created, you will be presented with the deployment details. Take note of the Instance ID.
8. Sign in to the AWS Management Console, select EC2 and click the ID for the instance just created. In the instance detail panel, make a note of the Public DNS hostname. You will need this to log in to your Couchbase Server installation.
9. To connect to your Couchbase Server installation, open a browser and connect to the instance URL on port 8091. For example, `http://ec2-107-21-64-139.compute-1.amazonaws.com:8091`. You will be prompted for the user name and password for the Couchbase Server web console:

   User name is *Administrator*
   Password is *your instance ID*

10. When you have successfully logged in, you should be presented with the Couchbase Server Administration Web Console Cluster Overview window. The server will have been automatically configured for you.

After the instance has started, it will operate just like any Couchbase Server instance. This single instance can operate independently, or can be added to a group of similar instances to create a cluster.

Manual Installation

To get started installing Couchbase Server manually, follow these steps:

1. Launch an Amazon EC2 instance using the AMI of your choice. (Our example will use the Amazon Linux 64-bit AMI).

2. Create an EBS volume to use for your Couchbase Server storage, and attach it to the instance.
3. Connect to the instance over SSH.
4. Install the openssl098e that couchbase server 2.0.1 requires.

   ```
   $ sudo yum install openssl098e
   ```

5. Download and install the Couchbase Server package. At the SSH command prompt, type the following, and then press Enter:

   ```
   $ sudo rpm -i couchbase-server-enterprise_x86_64_2.0.1.rpm
   ```

6. Couchbase Server will start automatically when the installation is complete. The setup process, which configures Couchbase Server and sets the data location and other details, is run separately.
7. Make a file system on your Amazon Elastic Block Store (EBS) volume:

   ```
   $ sudo mkfs -t ext4 /dev/*the_connection_you_attached_the_volume_to_for_example_sdf*
   ```

8. Make a directory to mount the file system to:

   ```
   $ sudo mkdir -p /data/db/
   ```

9. Edit your file system configuration to enumerate the volume on startup:

   ```
   $ sudo su
   $ sudo echo '/dev/sdf /data/db auto noatime,noexec,nodiratime 0 0' >> /etc/fstab
   $ exit
   ```

10. Mount the volume:

    ```
    $ sudo mount -a /dev/sdf /data/db
    ```

11. Set the owner of the mounted volume:

    ```
    $ sudo chown couchbase /data/db
    ```

12. Configure the instance to use the Amazon EBS directory for storage:

    ```
    $ /opt/couchbase/bin/couchbase-cli node-init -c localhost:8091 --node-init-data-
    path=/data/db
    ```

13. Set the security group for the new instance, allowing connectivity for ports 8091, 8092, 11210, 4369 and 21100 to 21199. For communication between clients and instances, you need only ports 8091, 8092, and 11210.
14. Initialize the cluster by setting the username, password, cluster port (default 8091) and RAM allocated to Couchbase Server for data storage. To do so, access the Couchbase Server setup service through the Web-based administration console, which is initially exposed on port 8091. If you have not configured a public IP address, or if you want to automate the process, you can use the command-line tools to perform the setup operation as follows:

    ```
    $ /opt/couchbase/bin/couchbase-cli cluster-init -c localhost:8091 \
       --cluster-init-username=Administrator \
       --cluster-init-password=password \
       --cluster-init-port=8091 \
       --cluster-init-ramsize=8000
    ```

15. When the cluster is initialized, you can access the Web Administration console by visiting the public DNS on the port configured in the command in the previous step, for example, http://publicdns:8091. You must log in with the user name and password configured in the previous step.

Architecture

The design of your Couchbase Server installation on Amazon EC2 depends mainly on the scale at which you're trying to operate. If you're experimenting with the framework on your own for a private project, we recommend creating a cluster with a minimum of three instances. Beyond three instances, the actual sizing will depend on the overall amount of data that you want to store and on your performance requirements.

As demand for your application increases, you can add instances to your cluster to cope with additional RAM or I/O loads. You can expand your Couchbase Server cluster without taking it offline by using the online rebalance operation as described in the Expanding Your Couchbase Server Cluster section. Your application stays online as you upgrade the cluster a few instances at a time.

Building blocks

In this section, we will discuss the pieces and concepts that makes up a Couchbase Server Cluster.

Buckets and vBuckets

Couchbase Server uses data buckets as logical containers of information. Aside from providing logical separation of information, individual buckets can also be password-protected and can support specific levels of replication. For example, you can configure one bucket with replicas for sensitive information and another bucket for session data where replicas would not be required.

Underlying the bucket structure is the vBucket system. vBuckets are an abstraction layer that allows the information in the bucket to be distributed and sharded across the different instances in the Couchbase Server cluster. Clients can use the vBucket map to determine which vBucket resides on which Couchbase Server instance. The vBucket map is also used during the rebalance operation to transfer data when you increase or decrease cluster size or after a failover.

Clients use information in the vBucket map to communicate directly with each cluster instance when the client is storing data. When the cluster configuration or size is changed during a rebalance operation, the vBucket map is updated and sent to each client, which uses the updated information to ensure that it stores and retrieves information directly with the right instance.

Replicas and Failover

Couchbase Server can optionally keep replicas of your data for use in the event of an instance failure. Replicas are used only for this purpose; they do not affect or relate to the distribution and scalability of your dataset. The replicas are configured on a bucket-by-bucket basis, and each bucket can have zero to three replicas. Data is stored in the replicas after it has been updated on the instance responsible for the document, but before the data has been permanently stored in the source bucket.

Replica data is also automatically sharded by vBucket: replica copies of the data are distributed across all the instances of the cluster in the same way that the original source data is distributed. Automatic sharding further enhances the resilience of the cluster, because even a multiple instance failure may be unlikely to make the stored data unavailable.

When an instance fails over because of a severe hardware or software issue, the replica vBuckets of the data stored on that instance are enabled, and clients will communicate directly with the replica instances instead of the original. To re-enable the failed instance once it has been repaired or replaced, you must perform a rebalance operation, which re-establishes the replica vBuckets and also redistributes data uniformly across the cluster.

Use of replicas is entirely optional, and there may be datasets and use cases where replicas are not required. For example, session data for your web application may not require replicas. By using different buckets for your data types and setting replica counts individually for each bucket, you can leverage granular control over your replica configuration and failover support, providing resilience for critical datasets, and minimizing the operating cost of disposable datasets.

Views

To provide support for indexing and querying, documents stored in Couchbase Server 2.0 can create one or more indexes of the data, called views. Each Couchbase data bucket can have one or more design documents, and each design document can define one or more view. To create each view Couchbase Server iterates over every data document in the bucket and then stores the generated index information as a view. Once the initial index has been created, only documents that have changed have to be processed in order to update the view.

Documents are processed for inclusion in the index for each vBucket across all of the configured instances within the cluster. The indexing process is therefore both distributed and supported during a rebalancing operation. As a result, queries can operate with a consistent view of the index data, even during a rebalance. Buckets can also be configured with replica indexes, which create a copy of the index data to be used during failover.

Documents are indexed only after they have been written to block storage, and items are deleted from the index only after they have been flushed both from RAM and from persistent data storage. To help ensure that your indexes are kept up to date, the performance of your storage system must keep pace with the rate of document change. A larger storage queue will lead to delays when you are indexing stored documents.

Cross Datacenter Replication (XDCR)

Cross Datacenter Replication (XDCR) enables replication of data from one cluster to another. XDCR can copy both existing information and changed documents. Replication is configured in one direction only. To configure bi-directional replication, you must configure replication from cluster A to cluster B, and from cluster B to cluster A. Built-in conflict resolution automatically identifies the most recent version of a document by comparing document data and metadata.

When you are configuring replication, whether between clusters within Amazon EC2, or between Amazon EC2 and non-Amazon EC2 clusters, care should be taken with the network configuration and port access. Information replicated between clusters is unencrypted. When replicating between AWS and non-AWS networks, the VPN functionality built into VPC can be leveraged for encryption. However, to encrypt your data during replication between AWS regions, you will need to use a third-party VPN solution.

Sizing

Measuring and predicting the size of your dataset is critical to understanding the optimum configuration for your cluster. There are some basic figures that you can use to help provide a rough estimate of your cluster size and configuration requirements:

- *Working set* – The amount of data that should be kept in memory for optimum performance. Data stored in RAM within Couchbase Server may be available with sub-millisecond access times; data that has to be loaded from storage may take substantially longer to retrieve. Your working set should be quantified as a percentage of your total document storage requirements.

- *Required RAM* – The amount of memory required to store all of your working set. To calculate required RAM, multiply the number of documents you expect to store by the average size of each document. You should also

separately calculate the size of the metadata required for your documents. Metadata is always kept in RAM, so you must have enough RAM to store the metadata and the 'working set' proportion of your document data. Exact metadata sizes depend on a number of variables, but using a figure of 150 bytes per document may provide a good baseline.

- *I/O operations* – An estimate of the number of changes per second you expect on your data set. Knowing the number of I/O operations will help to indicate whether your cluster is ultimately RAM or I/O bound.

For example, if you want to store 1,000,000 documents, each of 32 KiB in size:

- Total metadata size: 134 MiB (140*1,000,000/1024/1024)
- Total data size: 31 GiB (32768*1,000,000/1024/1024/1024)

To keep the entire dataset within RAM, you will need 32 GiB of RAM across the cluster.

For each replica configured, you must add that number again to keep it in memory. For example, two replicas will require 96 GiB of RAM (one original, and two replicas).

Using Extra Large instances with 9 GiB allocated to Couchbase Server, you would need twelve instances to keep all the data in RAM. If your working set is less than 100% of your total dataset size, it may be possible to reduce this allocation.

Next, you should calculate the number of I/O operations per second (IOPS) supported by your chosen storage solution. By dividing your expected updates per second by the IOPS of your EC2 instances, you can estimate the required size of your cluster. For example, using a single provisioned IOPS EBS volume on each instance in the cluster configured with 1,000 IOPS, 16,000 updates per second would require 16 instances. (Note: EBS reads are limited to 4KiB, so manipulating large items may take multiple I/O operations.)

Additional consideration should be taken to provide I/O for views and querying, if you are using them, and for both incoming and outgoing XDCR threads. Without careful monitoring, the exact requirements for view and XDCR overhead can be difficult to predict. Your cluster should be monitored and instances added or removed as needed.

RAM Configuration

Couchbase Server incorporates a built-in object level cache that operates as a core part of the overall system. Data is always read from and written to RAM. During a write operation, the data is updated in RAM first, and then the information is replicated to the other instances in the system (if replicas are enabled) and asynchronously written to persistent storage. During a read operation, the information is returned from RAM if it is available. If not, a background process loads the data from storage into RAM and then returns the data to the client.

Within Couchbase Server 2.0, the recommended maximum amount of RAM to be allocated to Couchbase Server for caching purposes is 60% of the physical RAM available. The additional overhead allows the operating system to cache file system data, which can improve the performance of views. Couchbase Server does not explicitly cache view index information; it does so only through the standard operating system storage cache.

Storage I/O

You should ensure that the provisioned size of the storage volumes for your data is sufficient to store all of the information for your application. All data within Couchbase Server is ultimately written to volume storage; therefore,

your storage space must be large enough for your RAM configuration and dataset, along with a buffer to allow for updates, fragmentation, and expansion of your dataset.

The view index information must be written to storage when the index is generated. In addition, querying of the indexes requires storage access to load the index information. The storage I/O overhead is therefore dependent on the quantity and complexity of your views and on the number of queries performed through the generated indexes.

To improve performance, during installation you can configure separate paths for storing document data and index data. To improve performance, you should use different EBS volumes for each path. You can configure separate paths for the data and index storage during provisioning and setup of each individual instance.

Both the document data and the view index information are written to storage in an append-only format. Changes to the document data and index updates create fragmentation on storage of the active data. A process called compaction rebuilds the stored data to reduce fragmentation. Compaction can occur any time while your cluster is running. It can be scheduled to occur during specific times and triggered at specific fragmentation levels.

XDCR also requires storage I/O as record changes are loaded from storage to be sent to the destination cluster, and any incoming streams of changes from another cluster must also be updated and written to storage. The larger the number of outgoing and incoming XDCR threads, the greater the I/O overhead.

Within AWS, the greatest limiting factor to the performance of Couchbase Server is often storage I/O. The following table outlines the expected I/O operations per second of the different storage types available on AWS.

Storage Type	IOPS
Standard EBS	~100
8-volume Striped EBS RAID	~800
Provisioned IOPS	Up to 4,000
8-volume Striped RAID Provisioned IOPS	Up to 32,000
HI1.4xlarge SSD Volumes (RAID0)	~80,000

To help prevent storage I/O from impairing performance, you need enough throughput to accommodate the updates that you will write to the data documents. After an update has been written to persistent storage, the data will remain in RAM. If the system is running out of memory but needs to store new information, it can free the memory that the stored update was occupying and allocate it to the new data.

If the available throughput is insufficient to keep up with write operations, the RAM cannot be reclaimed quickly enough, and new read attempts from data documents will cause an out of memory error until more memory becomes available. Your cluster should therefore be able to handle the required throughput with some headroom to allow for burst rates, and it should provide spare capacity to give you time to expand your cluster as your requirements grow.

To support the aggregate I/O required, we recommend that you increase the number of instances and, if appropriate, lower the RAM requirement on each instance. A higher instance count will give better overall storage performance and also provide better resilience in the event of a failure, as losing a single instance will have a reduced impact.

To improve I/O performance further, you can move from a single EBS volume to a RAID-based EBS volume. The recommended RAID solution is six to eight EBS volumes in a RAID0 configuration. For even better performance, use the provisioned IOPS EBS volumes, which provide a specific level of I/O performance. For the highest performance, use the High I/O Quadruple Extra Large EC2 instance (hi1.4xlarge), which uses SSD-based storage.

Storage and Index Compaction

Information is written to storage and indexes are updated by appending information to the data files. Where data has changed or been deleted, the space is not automatically reused. You can reclaim the space used by using compaction. Compaction copies the active data to new document files and then deletes the originals. Compaction can be performed manually, or you can configure automatic compaction. Automatic compaction supports both clusterwide and bucket-specific triggers, and the timing and operation of the compaction process can be individually controlled.

Using compaction implies additional storage I/O and CPU overhead, but the process can be completed while your cluster is still running and servicing clients. You can control both the fragmentation level and the times when compaction takes place to optimize the running time. If you have the necessary I/O and CPU capacity, compaction of the document and index files can take place simultaneously. For best performance, you should place the data documents and index files on separate logical volumes, which will make it easier for the system to cope with simultaneous compaction.

CPU Requirements

Couchbase Server is not typically CPU-bound for pure key/value workloads, where RAM and storage I/O are more operationally significant. A configuration of at least 4 cores may provide enough CPU resources to handle reading and writing of data and background functionality, such as sharding and replica support. During a rebalance operation, there is a CPU usage increase as the data is moved about the instances within the network.

Using views and XDCR implies additional CPU processing overhead to process the documents and build the feeds that transfer data between clusters. If your deployment uses views, XDCR, or both, you should use instances with higher ECU values.

Network Configuration

For communication between instances within the cluster, you must have opened the appropriate ports for communication by using a suitable security group. For the IP addressing, the ideal mode will depend on how your clients will be accessing the database. If the clients are accessing Couchbase from within Amazon EC2, then use the private IP address for each instance to permit the instances to communicate within the Amazon EC2 network.

If your clients are outside Amazon EC2, you must configure each Amazon EC2 instance with a public IP address and then use this IP address to identify and add each instance to the cluster. Couchbase Server exchanges the cluster map by using the registered IP address. You can combine the public IP address with a DNS solution such as Amazon Route 53 to provide a convenient name for your cluster instances.

For administration, particularly if you want to use the Web Administration Console, you will need to use the public DNS address or enable a public IP address on one or more instances within your cluster. You must also ensure that your security group is configured to allow communication on the administration port (8091) so the administration client can communicate with your EC2 instances and all EC2 instances within the cluster can communicate with each other.

Network Configuration for XDCR

If you are using Couchbase Server in combination with XDCR, there are additional requirements for communication within the cluster. XDCR operates by transferring information between individual instances within each cluster on port 8091 and port 8092. For a cluster that is configured for XDCR, the security group associated with every instance in every Couchbase Server cluster must open these ports to all the instances in every Couchbase Server cluster.

The data transferred between clusters with XDCR is unencrypted. To help secure the replicated information, you will need to configure a suitable VPN gateway between the two data centers that will encrypt the data between each route between data centers. Configuration of these VPN routes and systems is dependent on your VPN solution.

Production Designs

In this section, we will identify common best practices for designing production Couchbase Server implementation.

Given the building blocks discussed in the previous section, how would a system scale to accommodate growing load over time?

Couchbase Server differs from many other structured storage systems by accommodating easy scalability by adding nodes. Because of the high-availability benefits associated with replication on multi-node systems, we recommend multi-node systems for all production applications, even though single-node deployments of Couchbase Server can work well.

Minimum Production scale: EBS-optimized m1.large instancesare the first, smallest scale choice for a Couchbase Server cluster.

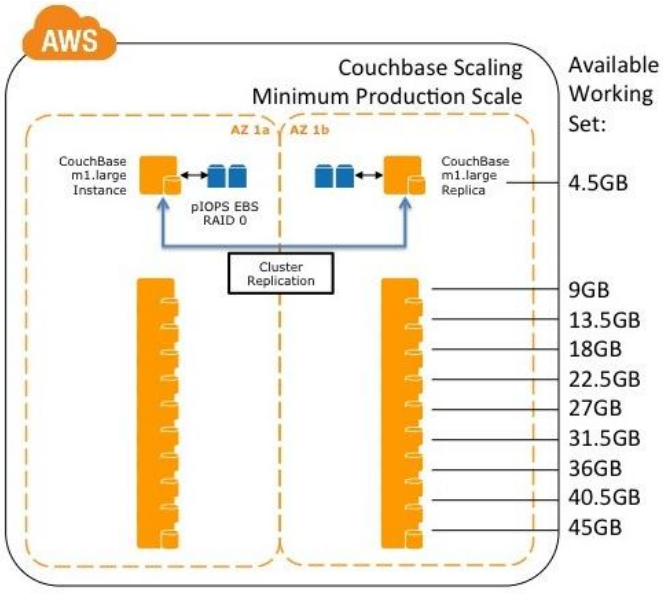

Figure 1: Minimum Production Scale - Couchbase Architecture on AWS

You may choose this setup when your working set is unlikely to grow faster than a few GiB a month. Starting with an initial cluster of two instances configured with one replica, it's easy to add highly available capacity of about 4.5 GiB at a time by adding two additional m1.large instances. (Remember that Couchbase Server should use only 60% of available memory.) With a 20-instance m1.large cluster that uses one replica providing approximately 45 GiB of high-performance working-set capacity and 5 Gbps access to EBS storage, your initial cluster can be grown in small steps according to your actual growth rate. If the workload is write-heavy, a RAID0 configuration of two pIOPS EBS volumes with 4,000 IOPS on each instance would provide up to 80,000 IOPS of consistent write performance across the cluster.

Medium scale: For larger-scale deployments, where the working dataset starts above 40 GiB or is likely to grow by more than 10 GiB a month, using m2.4xlarge EBS-optimized instances will provide a more manageable solution.

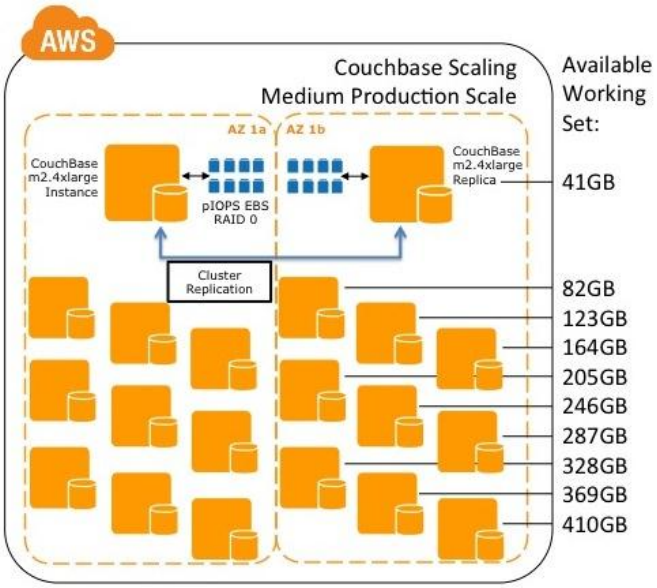

Figure 2: Medium Production Scale: Couchbase Architecture on AWS

Following the same pattern above but using bigger basic building blocks, a 20-instance cluster that uses one replica would provide approximately 410 GiB of high-performance working-set capacity and 10 Gbps access to EBS storage. Similarly, adding a RAID0 configuration of eight Provisioned IOPS EBS volumes at 4,000 IOPS each to each instance would provide an aggregate of 640,000 IOPS of consistent write performance across the cluster.

Large scale, write-oriented: For the largest write-oriented workloads, where the full dataset will always significantly exceed available memory capacity, hi1.4xlarge instances and their 2 TiB of local ephemeral SSD storage can provide exceptional performance.

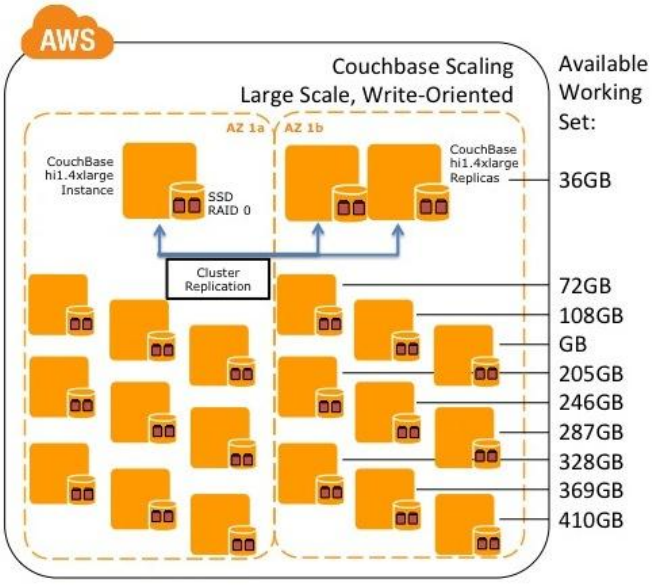

Figure 3: Large Scale, Write-Oriented: Couchbase Architecture on AWS

Because this solution uses ephemeral storage, we recommend a minimum of two replicas, as well as aggressive use of the cbbackup tool, to help improve disaster recovery. In this case, a 30-instance cluster using two replicas would provide approximately 360 GiB of high-performance working-set capacity and over 10 GBps (gigabytes, not gigabits) access to storage, for approximately 500,000 IOPS of write performance across the cluster.

Extra Large scale, read-oriented: For workloads where performance requirements force the entire large-scale dataset to fit into working memory, cr1.8xlarge instances may offer better performance.

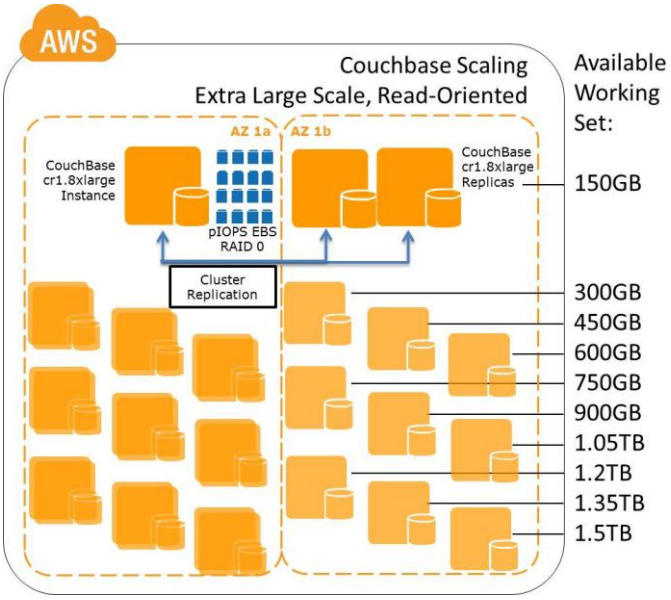

Figure 4: Extra Large Scale, Read-Oriented: Couchbase Architecture on AWS

Because this solution uses ephemeral storage, we recommend a minimum of two replicas, as well as aggressive use of the cbbackup tool, to help improve disaster recovery. In this case, a 30-instance cluster using two replicas would provide approximately 1.5 TiB of high-performance working-set capacity, with the option of using local ephemeral SSD to deliver about 60,000 IOPS of storage performance, or 20 Gbps of access to EBS Provisioned IOPS storage, delivering up to 640,000 IOPS of consistent write performance.

As with all scaling solutions with Couchbase Server, adding more instances to the cluster will help extend the performance.

Expanding your Couchbase Server Cluster

Expanding the number of instances within your cluster is the primary method for increasing the overall performance of your Couchbase Server cluster. Whenever you add or remove instances, you must perform a rebalancing operation. Rebalancing moves the data around the cluster to match the new instance layout and reduce the load on individual instances so that each instance is handling the optimum amount of data.

You can add and remove multiple instances to the cluster in one operation. In general, it is better to add multiple instances to your cluster and perform a single rebalance operation than to add each instance individually and performing a rebalance operation each time.

To add instances to an existing cluster:

1. Start new instances that you can add to your Couchbase Server cluster. You may want to create new instances with a different EBS configuration or instances that use SSD storage.

2. Set the security group for the new instances to confirm that the port configuration is correct for your Couchbase Server installation.

3. For hi1.4xlarge or cr1.8xlarge instances, you must set the data location individually on each new instance to confirm that data is stored on the EBS volume or local SSD. You can set this configuration by using either the Web based UI during the setup phase, or at any time by using the command-line tool couchbase-cli:

```
$ /opt/couchbase/bin/couchbase-cli node-init -c instanceip:8091 --node-init-data-path=/mnt/ebs
```

You should run this command with the appropriate IP address (public or private) for each instance that you are adding to the cluster.

4. Add the instances to the cluster and then perform a rebalance by using either the Web UI or the couchbase-cli command. You can specify all the instances using just a single command:

```
$ /opt/couchbase/bin/couchbase-cli rebalance -c clusterip:8091 \
   -u Administrator -p password \
   --server-add=instanceaip:8091 \
   --server-add=instancebip:8091 \
   --server-add=instancecip:8091
```

Where clusterip is the IP address of an existing instance in the cluster, the Administrator and password are the username and password of the administrator of the cluster, and the instanceaip, instancebip, and instancecip are the IP addresses of each new instance that are to be added to the cluster.

All the specified instances will be added to the cluster, and a rebalance operation will be started. Progress information will be displayed on the console. You can also add, rebalance, and monitor the rebalance operation by using the Couchbase Administration web console.

Once the rebalance operation has been completed, you should monitor the status and health of your cluster and determine if you need additional instances.

Expanding your Couchbase Server Cluster using AWS CloudFormation

For simpler multiple deployments of Couchbase Server instances, there is a suite of AWS CloudFormation templates that create multiple instances that can be automatically configured and attached to your server.

To use the templates, download the Couchbase cloud-formation project from Github:

```
$ git clone git://github.com/couchbaselabs/cloud-formation.git
```

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation/. Click **Create New Stack**.

2. Fill in a stack name.

3. Upload a template file from the templates that you downloaded, choosing a template appropriate for the size of your cluster. For this example, we will use the 64-bit 1.8.1 template.

4. Specify the parameters for the new instances:

   `RESTPassword` is the administrator password for the cluster.
   `RAMforDefaultBucket` is the amount of RAM to be allocated to the default Couchbase Server bucket.
   `CouchbasePackageURL` is the location of the Couchbase Server package.
   `UserTag` is a unique identifier to help you identify the cluster.
   `ExistingClusterHostPort` is the host IP and port number of an existing Couchbase Server cluster.
   `RAMPerServer` is the amount of RAM configured for each instance within the cluster.
   `InstanceType` is the EC2 instance type to be used for the template.
   `KeyName` is the Secure Shell (SSH) key to be used for each cluster instance.

For new clusters, you can provide values for KeyName, RESTPassword and RAMPerServer that are appropriate to the instance type that you have chosen. When adding a group of servers to a new cluster, supply the ExistingClusterHostPort parameter with the IP address and port number of an existing cluster instance. The new instances will automatically be added to the cluster. You must still trigger a rebalance operation to enable the new instances on the cluster.

Shrinking your Couchbase Server Cluster

You should reduce the size of your Couchbase Server cluster only when you are sure that your current capacity is more than you need. Removing an instance and rebalancing will place a significant increase in load on the remaining servers within the cluster. Because of the potential effect on the remaining instances, we recommend that you remove only a single instance at a time perform a rebalance, and then monitor cluster performance and health before removing further instances.

To remove a single instance from the cluster:

Mark an instance from removal by using the Web management client: under the **Management->Server Instances** section, select the **Remove** check box for the server that you want to remove, and then click **Rebalance** to the start the rebalance operation.

From the command line, use the couchbase-cli tool to remove the instance, and then start the rebalance operation:

```
$ /opt/couchbase/bin/couchbase-cli rebalance -c clusterip:8091 \
  --server-remove=instanceip
```

The rebalance operation will be started and the data on the instances will be redistributed according to the new configuration. Only when the rebalance operation has completed successfully can you stop and terminate the removed instance. Until the rebalance is complete, the instance will still be servicing requests from your clients.

Cross Datacenter Replication (XDCR) Patterns

In this section we will discuss patterns for Cross-Datacenter Replication (XDCR) to enable fault tolerance and disaster recovery.

With XDCR you can start, stop, and restart replication, so you can create a snapshot of data that exists in a cluster in a particular moment. The result is that there are a number of possible solutions you can implement with XDCR:

- **Creating an Active Offsite Backup** You can use XDCR to provide a live backup of your application data in a separate cluster. This secondary cluster can be local or in a different AWS region. If the main cluster fails, you can either enable the secondary cluster or use the data stored in the secondary cluster to repopulate the data in the primary cluster.
  For example, you could use an active offsite backup to configure your infrastructure as follows:
    - Cluster A is your primary cluster. It holds your live data and is actively used by your application servers to support your application.
    - You set up unidirectional replication to replicate data from Cluster A to a backup cluster, Cluster B.
    - When a failure occurs, Cluster A needs to be re-created when the cluster is functioning once again. You can then use unidirectional replication to replicate the data stored on Cluster B back to Cluster A, and then re-enable your application on Cluster A.

- **Spreading Cluster Data Geographically** You can use bidirectional replication to synchronize data between Couchbase Server buckets in two different clusters, including clusters in different locations. You can use this model to spread the load geographically, for example, to spread data between the US-EAST-1 and EU-WEST-1 regions, while maintaining a consistent data set across clusters. In this case, you would configure your clusters as follows:

    - Configure Cluster A to replicate data from one or more source buckets to destination buckets on Cluster B.

    - Configure Cluster B to replicate data from the destination buckets to source buckets on Cluster A.

    - XDCR replicates any changes to data stored on Cluster A to destination buckets on Cluster B and vice versa. This action keeps the two clusters in synchronization with each other.

Views, Indexing, and Querying Best Practices

In this section, we will discuss best practices for views, indexing, and querying.

Views and indexes should be written according to the needs of your data, application, and querying requirements; however, care should be taken, because the view structure and complexity can affect execution and deployment, particularly with respect to the storage I/O and CPU performance. These effects include, but are not limited to, the following:

- **Quantity of Views per Design Document** Because the index for each map/reduce combination within each view within a given design document is updated at the same time, avoid declaring too many views within the same design document. For example, if you have a design document with five different views, all five views will be updated simultaneously, even if only one of the views is accessed. Updating all the views simultaneously can result in increase view index generation times, especially for frequently accessed views.

  Instead, move frequently used views to a separate design document. The exact number of views per design document should be determined from a combination of the update frequency requirements on the included views and grouping of the view definitions. For example, if you have a view that needs to be updated frequently (for example, comments on a blog post), and another view that needs to be updated less frequently (for example, top blogposts), separate the views into two design documents so that the comments view can be updated independently of the top blogposts view.

- **Modifying Existing Views** If you modify an existing view definition or are executing a full build on a development view, the entire view will need to be re-created. In addition, all the views defined within the same design document will be re-created. Rebuilding all the views within a single design document is expensive in terms of I/O and CPU requirements, as each document will need to be parsed by each view's map() and reduce() functions, with the resulting index written to storage. This process of rebuilding will occur across all the instances within the cluster, which increases the overall storage I/O and CPU requirements until the view has been re-created. This process will take place in addition to any production design documents and views that also need to be kept up to date.

- **View Size, Storage and I/O** Within the map function, the information declared within your emit() statement is included in the view index data and then written to storage. Outputting this information will have the following effects on your indexes:

- **Increased index size on storage volumes**– More detailed or complex key/value combinations in generated views will result in more information being persisted.

- **Increased volume I/O** – A large, complex key/value definition in your view will increase the overall volume I/O required both to update the data and to read the data back. As a result, the index can be quite large. In some cases, the size of the index can exceed the size of the original source data by a significant factor if multiple views are created, or you include large portions or all of a data document in the view output. For example, if each view contains the entire document as part of the value and you define ten views, the size of your index files will be more than 10 times the size of the original data on which the view was created. With a 500-byte document and 1 million documents, the view index would be approximately 5 GB with only 500 MB of source data.

- **Don't Include Entire Documents in View output** A view index should be designed to provide base information through the implicitly returned document ID point to the source document. Including the entire document within your view output can severely affect performance. You can always access the full data document through the client libraries by requesting the individual document. This way is typically much faster than including the full document data in the view index, and it enables you to optimize the index performance without sacrificing the ability to load the full data document.

- **Use Built-in Reduce Functions** Where possible, use one the supplied built-in reduce functions, _sum, _count, or _stats. These functions are highly optimized. Using a custom reduce function requires additional processing and may impose additional build time on the production of the index.

Anti-patterns

In this section, we will identify practices that are generally best to avoid.

1. Creating a large cluster far in excess of either your RAM or storage I/O requirements will provide you with no significant benefits except the ability to grow into your cluster without having to add instances and rebalance. Instead, construct a cluster that can cope with your known data requirements with suitable headroom for expansion, and then increase the cluster as required before it becomes overloaded.

2. Rebalancing places additional memory and I/O load on your cluster because it involves physical data movement across instances in the cluster. As a result, you should aim to increase the size of your cluster before you reach cluster capacity. Keeping the equivalent RAM and I/O capacity of one or two instances in your cluster as a buffer before you upgrade is good practice, and it will enable you to expand your cluster before you run out of capacity. The rebalance operation should be run only when the cluster is in a healthy state. A good metric to watch out for is the size of the storage write queue across the cluster. If even a single instance has a storage write queue above 1 million documents, you should wait to perform the rebalance operation.

3. All instances within the cluster are treated equally, so there is no reason to have different RAM or storage configurations on each instance. A better model is to choose a RAM configuration and stick to it. You can change the underlying storage configuration over time by swapping instances with different storage setups. The RAM configuration can also be modified over time if you move the entire cluster to instances with more memory.

4. Avoid using instance types where the I/O performance is low or medium. To provide high I/O performance to EBS volumes of any type, use an instance with high I/O performance and support for EBS optimization.

Operations

In this section, we will discuss ongoing maintenance and cluster administration tasks.

Rebalancing

Whenever the configuration of your cluster changes—for example, you add or remove an instance, an instance on the cluster fails, or a failed instance is brought back online—a rebalance operation will redistribute the data to reflect the new instance layout and configuration.

Rebalancing can be performed through the Web Console, through the REST API, or by using the command-line tools. Rebalancing proceeds while the cluster is running, so it can have a small impact on performance as data is moved between instances.

Couchbase Server gives active instance operations a higher priority than the rebalance operation. If the cluster is busy, the rebalance operation may take a long time.

To start a rebalance operation from the command-line, use the couchbase-cli command:

```
$ couchbase-cli rebalance -c clusterip:8091
```

The rebalance progress will be displayed onscreen.

Compaction

Compaction of the data and index files reclaims the space used to write information to storage. The compaction process can occur while your cluster is running and your application is still processing requests. The compaction process takes the following steps:

1. Creates a new file for the stored data.
2. Copies the active information from the existing files to the new files.
3. Enables the new files as the active storage data and then deletes the previous versions.

Because the compaction process operates by creating new files on storage containing the active information, there must be double the amount of storage space used by the files for the compaction process to complete effectively.

Couchbase Server incorporates an automated compaction mechanism that can compact both data files and the view index files whenever the current fragmentation level within the database and view index data files exceeds threshold that you specify.

> **Note**
>
> Spatial indexes must be compacted manually. For more information, see Section 5.4.4, "Compacting Spatial Views," in the Couchbase Server Manual.

Auto-compaction can be configured in two ways:

- **Default Auto-Compaction** affects all the Couchbase buckets within your Couchbase Server cluster unless you override it for specific buckets. If you set the default auto-compaction settings for your Couchbase server, then auto-compaction is enabled for all Couchbase buckets.

- **Bucket Auto-Compaction** can be set on individual Couchbase buckets. Bucket-level compaction always overrides default auto-compaction settings. If you have not configured default auto-compaction, you can still configure bucket auto-compaction on specific buckets.

The available settings for both default auto-compaction and Couchbase bucket-specific settings are identical:

- **Database Fragmentation** The primary setting is the level of fragmentation within the database at which compaction occurs. The figure is expressed as a percentage of fragmentation for each item, and you can set the fragmentation level at which the compaction process will be triggered. For example, if you set the fragmentation percentage at 10%, the moment the fragmentation level is reached for any item, the compaction process will start unless you have limited auto-compaction to a time period as described later in this list.

- **View Fragmentation** View fragmentation specifies the percentage of fragmentation within all the view index files at which compaction will be triggered.

- **Time Period** To prevent auto-compaction when your database is in heavy use, you can configure a time during which compaction is allowed, expressed as start and end times that are specified to the minute. For example, you could configure compaction to take place between 01:00 and 06:00. If the threshold for compaction is met outside of these hours, compaction will be delayed until the specified time period.

  > **Note** The same time period is observed every day while the Couchbase Server is active. The time period cannot be configured on a day-by-day basis.

- **Compaction abortion** If you specify a time period, it is possible that the compaction process is in progress when the time period ends. You can specify how Couchbase Server proceeds by setting a compaction abortion option:

  - **Enabled** If compaction is running at the end of the specified time period, the compaction process will be stopped. Files generated during compaction will be kept, and compaction will be restarted at the beginning of the next time period. If you are very sensitive to system performance outside the time period, enabling compaction abortion will help to avoid the compaction process unexpectedly interfering with other database activities.

  - **Disabled** If compaction is running at the end of the specified time period, it will continue to completion.

- **Parallel Compaction** By default, if both the database and the views are configured for auto-compaction, compaction runs first on the database and then on the views. If you enable parallel compaction, both the databases and the views can be compacted at the same time. Parallel compaction requires more CPU and database activity, but if you have sufficient CPU cores and storage I/O (for example, if the database and view index are stored on different storage devices), compaction will be much faster.

Backing Up Your Couchbase Server Data

You should back up your Couchbase Server data frequently. Backups can be performed on a live and running system, but the method and frequency you use will depend on the quantity and rate of updates, and the data availability required. Because replicas and failover support within a Couchbase Server cluster help to support online availability, data backups should be used mainly for long-term data retention and disaster recovery.

Backup using cbbackup

We strongly recommend the use of the supplied cbbackup tool. This tool reads information from the Couchbase Server cluster and creates a copy of stored information. The backup can be performed across the entire cluster, a bucket across the entire cluster, individual instances, and individual buckets. All backups can be executed while the cluster is running. Each backup will require at least as much as storage space on a volume attached to the host performing the backup as is currently being used by your Couchbase Server to store data.

In order to perform a backup of the entire cluster and all buckets:

1. Create a new EBS volume to hold the backup data, and then attach it to your Amazon EC2 instance.
2. Run cbbackup, providing the user name and password of the administrator:

```
$ cbbackup http://HOST:8091 /backups/backup-20120501  -u Administrator -p password
```

The HOST can be any instance within the cluster. The backup created with cbbackup in this way can be restored to any cluster.

To back up only a single bucket, specify the bucket name with the -b parameter. For example:

```
$ cbbackup http://HOST:8091 /backups/backup-20120501 -u Administrator -p password -b default
```

Note: Because you perform a backup on a running system, it is logically impossible to create a backup that guarantees a complete copy of all the content stored in the database at a single point in time without suspending write operations. To handle an instance failover, you should configure one or more replicas for your bucket, rather than restoring backup data.

Restore using cbrestore

If you have backed up your data by using cbbackup, you can restore that backup, either to the same bucket, a different bucket, or to a cluster of a different size and configuration. The cbrestore tool restores data on a single bucket at a time; if your backup contains multiple buckets, you must restore each bucket individually.

For example, to restore a single bucket, recipes, from the backup data:

```
$ /opt/couchbase/bin/cbrestore /backups/backup-2012-05-10 http://Administrator:password@HOST:8091 --bucket-source=recipes
```

To restore the bucket to a different bucket on the destination cluster:

```
$ /opt/couchbase/bin/cbrestore /backups/backup-2012-05-10 http://Administrator:password@HOST:8091
    --bucket-source=recipes --bucket-destination=cookbook
```

Backup up using EBS Snapshot

You can use the EBS snapshot system to back up your running Couchbase Server cluster. To provide a consistent, usable snapshot, you must temporarily stop data being written to storage so that the files on storage are not being actively updated. You can then create the snapshot with the files in a known state. Recovery can be completed by restoring the data from the snapshot and then restarting the Couchbase Server instance.

As with the supplied cbbackup based solution, the EBS snapshot must be completed on each instance within the cluster. To create an EBS snapshot backup:

1. Stop persistence by running cbepctl on each bucket on each instance. This step will stop updates to the data from being added to the write queue. You can run cbepctl on a running cluster; however, a failure in the cluster during this process may lead to data loss, as data will not be written to storage until the snapshot process is complete.

   ```
   $ /opt/couchbase/bin/cbepctl host:port stop BUCKETNAME
   ```

   To determine when all information has been written to storage, check the ep_uncommited statistic on each instance. The value should be zero:

   ```
   $ watch -d '/opt/couchbase/bin/cbstats localhost:11210 all BUCKETNAME | grep ep_uncommitted'
   ```

2. On each instance, create a snapshot of the configured EBS data volume:

   $ ec-create snapshot –K key –C certificate VOLNAME –d "backup_instance1"

3. You should check the snapshot status to confirm that it has been captured properly:

   $ ec-describe-snapshots –K key –C certificate "backup_instance1"

4. Re-enable persistence on the instance so that data is written to the data files again:

   $ /opt/couchbase/bin/cbepctl host:port start BUCKETNAME

   This step must be executed for each bucket on each instance in the cluster.

Once the snapshots have been completed, they can be copied or archived where needed.

Restore Using EBS Snapshot

To restore an instance from an EBS snapshot, Couchbase Server must be shut down. The process is designed to be used when there has been a significant failure in your system and you need to bring the instance and cluster back to the most recent snapshot status. You are, in effect, re-creating the volume and data status at the time of the snapshot and starting Couchbase Server with the newly restored data files.

To restore from an EBS snapshot:

1. Create a new instance, or use an existing instance, and detach the existing volume used for data storage.
2. Create a new volume from your EBS snapshot:

   $ ec2-create-volume -K key -C certificate -s 70 -z us-east-1c --snapshot "backup_instance1"

3. Attach a new volume created from the EBS snapshot in the previous step:

   $ ec2-attach-volume -K key -C certificate VOLNAME -i INSTANCEID-d /dev/sdf

4. Start Couchbase Server:

   $ sudo /etc/init.d/couchbase-server start

   You can watch the warmup process, where data is loaded into memory from the storage content, by looking at the relevant warmup statistics:

   $ watch -d "/opt/couchbase/bin/cbstats localhost:11210 all BUCKETNAME| grep warm"

The process must be repeated on all Couchbase Cluster instances. Once all the instances have loaded the data from storage and are running normally, the cluster is ready to be used as normal.

Monitoring

In this section, we will discuss best practices for monitoring your Couchbase cluster.

AWS provides robust monitoring of EC2 instances, EBS volumes, and other services through the Amazon CloudWatch service. Amazon CloudWatch can send an alarm, by either SMS or email, when a user-defined threshold is reached. A great example would be setting an alarm on excessive storage throughput Another approach would be to write a custom metric to CloudWatch, for example, current free memory on your instances, and to trigger an alarm or an automatic response when the specified threshold is exceeded.

Couchbase Server includes an extensive statistic and monitoring facility, which is available through the command line, web console, and the Query API. There are hundreds of statistics that you can view and monitor to get the best information about the performance of your system. The following topics describe some key statistics that you can use to monitor your cluster. You can use this information as the basis for expanding or shrinking your cluster or for changing the storage configuration to suit your workload.

Monitoring RAM Usage

The overall RAM of your Couchbase cluster is the most important aspect that you should monitor. If you run out of RAM, Couchbase Server will start ejecting items from memory to make room for more recently used information. In a busy system, the effect is that more data is read from storage, which can have a cascading effect on storage write performance.

Watch for overall RAM usage, cache hit rates, and storage reads (which indicate that you are loading information from storage instead of providing the data from RAM). Using this information, you can also determine how the data is being used and written to storage, and you can predict how changes to the configurations (such as the instance changes) will affect performance.

To keep track of the memory used and memory available, Couchbase Server uses two watermarks to determine when information should be ejected from RAM to make space for new data. Only data that has been successfully written to storage is ejected.

- Low WaterMark - Replica data is ejected from memory when the memory level reaches this value.
- High WaterMark - The system will start ejecting active values out of memory when this watermark is reached. Once data has been ejected, requests for these items will need to be fetched from storage before being returned to the client.

If your system exceeds the high watermark, you may be in danger of running out of RAM. You should consider expanding your cluster to ensure all your data is stored and available in RAM.

You can monitor RAM usage and the effect on your cluster using the following statistics:

- Memory Used - The current size of memory used. If the memory used has reached a specified quota, then client requests will fail. Temporary errors indicate that the client can retry the operation later. If clients are receiving this error, that is one indication that your cluster needs to be expanded to support the additional load.
- Temp OOM errors per sec - The number of temporary out-of-memory errors per second should be 0. A higher value indicates a temporary lack of available memory. This condition may highlight a temporary spike in activity that the cluster is unable to cope with, and the cluster should be monitored to ensure that the condition is only transient.
- OOM errors per sec - The number of out-of-memory errors per second should be 0. A higher value means the cluster has run out of space to store information.
- Cache misses – The number of cache misses should ideally be no more than 10 percent of total requests, and it should never exceed 50 percent. Increasing values mean that data your application is requesting is not in RAM and must be loaded from storage before being returned to the client.

Problems in any of these statistics are an indication that your cluster is running out of RAM. Couchbase Server will not stop servicing requests, but performance will be degraded. Expanding the cluster by adding more instances will increase the available RAM and improve performance.

Monitoring Storage Usage

You can use the following statistics, available in the Couchbase Cluster Overview, to determine how efficiently the cluster is writing information to storage and whether the cluster can cope with the current frequency of changes.

Failure to effectively write information to storage can affect availability if one or more instances fail.

- Storage Write Queue Size – The size of the queue for data to be written to storage. This value should be compared within the configuration and size of the cluster and the level of changes. High values are not necessarily bad, as long as the cluster can sustain the writes to storage. An increasing value, however, indicates that your cluster is unable to support the current update frequency. Adding more instances to the cluster will increase the aggregate I/O and alleviate the problem. Alternatively, changing your underlying storage from EBS standard storage EBS pIOPS, and beyond that to EBS pIOPS RAID or SSD will further improve the I/O performance.

- Storage Fetches Per Second – The frequency that Couchbase Server is reading request values from storage instead of from RAM. In a deployment where your entire dataset is expected to fit in memory, this value should remain at zero.

- Storage Utilization – The overall amount of storage that is being used by your data and indexes. This information can help you determine whether you are running out of physical storage. Adding more instances with larger EBS configurations will increase your available storage, but you must make this change across all instances in the cluster, because the data is distributed equally across each instance.

Monitoring Network Performance

Network bandwidth within AWS is shared by the client, communication between cluster instances (particularly during replication), and storage I/O if you are using EBS. The performance of your cluster is affected by overall network performance and available network bandwidth. Each EC2 instance is provisioned with a non-configurable network bandwidth capacity, and so, if your cluster has too few EC2 instances, you may reach network utilization limits that further reduce the performance of your cluster. Adding further instances in this case will expand the available network bandwidth.

Monitoring XDCR Performance

XDCR statistics are monitored on both inbound and outbound replication operations. For each configured XDCR data stream (inbound or outbound), you can monitor in detail the replication process. For outgoing XDCR connections, you can monitor the number of documents to be replicated, the number successfully replicated, and the size of the queue for replication. For inbound replication, you can monitor the type and number of operations pending from each XDCR source.

Security

In this section, we will discuss some best practices regarding the security of your Couchbase Server cluster.

Network

Your application will need to communicate with Couchbase Server on port 11210 (for data), port 8092 to access the Views system, and port 8091 for obtaining cluster communication. For administration, only port 8091 is required. To enable self-administration, you must open port 8091 within the security group for self-reference (localhost).

For communication between instances within the cluster, you will need to open ports 8091, 8092, 4369, 11210 and 21100 to 21199 (inclusive).

Because you must keep both the data and administration ports open for your application servers, you should configure the Couchbase security groups to allow access to those ports from the security groups for your application servers. All other ports should be restricted.

Authentication

Authentication is not required for client applications to access individual buckets, but you can set a password on each bucket that the client must supply in order to connect to the bucket. For administration, including using the Query API, you must use the user name and password that were configured during setup.

Conclusion

AWS provides a unique set of services for running and managing NoSQL applications, including Couchbase. Pairing Couchbase's ability to expand a cluster without taking it offline, and EC2's elastic nature provide users the ability to build high performance, highly expandable NoSQL databases without needing to manage physical hardware. This whitepaper has provided an overview of best practices for implementing Couchbase on Amazon EC2, paying particular attention to performance, durability, and security.

Further Reading and References

1. The Couchbase Server 2.0 Manual is available online at http://www.couchbase.com/docs/couchbase-manual-2.0/index.html.

2. For specific cloud deployment and best practices, go to http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-bestpractice.html.

3. Background and architecture information on Couchbase Server is available at http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-introduction.html.

4. For information on specification administration tasks, including backups, XDCR and configuration, go to http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-admin-tasks.html.

5. Views, indexes, querying, and writing and deploying views and design documents are covered in detail in the views documentation at http://www.couchbase.com/docs/couchbase-manual-2.0/couchbase-views.html.