银河麒麟服务器操作系统 V4

CouchDB 软件适配手册

天津麒麟信息技术有限公司

2019 年 5 月

目　录

1 概述

1.1 系统概述

银河麒麟服务器操作系统主要面向军队综合电子信息系统、金融系统以及电力系统等国家关键行业的服务器应用领域，突出高安全性、高可用性、高效数据处理、虚拟化等关键技术优势，针对关键业务构建的丰富高效、安全可靠的功能特性，兼容适配长城、联想、浪潮、华为、曙光等国内主流厂商的服务器整机产品，以及达梦、金仓、神通、南大通用等主要国产数据库和中创、金蝶、东方通等国产中间件，满足虚拟化、云计算和大数据时代，服务器业务对操作系统在性能、安全性及可扩展性等方面的需求，是一款具有高安全、高可用、高可靠、高性能的自主可控服务器操作系统。

1.2 环境概述

服务器型号	长城信安擎天 DF720 服务器
CPU 类型	飞腾 2000+处理器
操作系统版本	Kylin-4.0.2-server-sp2-2000-19050910.Z1
内核版本	4.4.131
CouchDB 版本	1.6.0

1.3 CouchDB 软件简介

CouchDB 是用 Erlang 开发的面向文档的数据库系统。CouchDB 不是一个传统的关系数据库，而是面向文档的数据库，其数据存储方式有点类似 lucene 的 index 文件格式，CouchDB 最大的意义在于它是一个面向 web 应用的新一代存储系统，事实上，CouchDB 的口号就是：下一代的 Web 应用存储系统。

CouchDB 是分布式的数据库，他可以把存储系统分布到 n 台物理的节点上面，并且很好的协调和同步节点之间的数据读写一致性。这当然也得靠 Erlang 无与伦比的并发特性才能做到。对于基于 web 的大规模应用文档应用，分布式可以让它不必像传统的关系数据库那样分库拆表，在应用代码层进行大量的改动。

CouchDB 是面向文档的数据库，存储半结构化的数据，比较类似 lucene 的 index 结构，特别适合存储文档，因此很适合 CMS，电话本，地址本等应用，在这些应用场合，文档数据库要比关系数据库更加方便，性能更好。

CouchDB 支持 REST API，可以让用户使用 JavaScript 来操作 CouchDB 数据库，也可以用 JavaScript 编写查询语句，我们可以想像一下，用 AJAX 技术结合 CouchDB 开发出来的 CMS 系统会是多么的简单和方便。

1.4 工作原理

CouchDB 构建在强大的 B-树储存引擎之上。这种引擎负责对 CouchDB 中的数据进行排序，并提供一种能够在对数均摊时间内执行搜索、插入和删除操作的机制。CouchDB 将这个引擎用于所有内部数据、文档和视图。

因为 CouchDB 数据库的结构独立于模式，所以它依赖于使用视图创建文档之间的任意关系，以及提供聚合和报告特性。使用 Map/Reduce 计算这些视图的结果，Map/Reduce 是一种使用分布式计算来处理和生成大型数据集的模型。Map/Reduce 模型由 Google 引入，可分为 Map 和 Reduce 两个步骤。在 Map 步骤中，由主节点接收文档并将问题划分为多个子问题。然后将这些子问题发布给工作节点，由它处理后再将结果返回给主节点。在 Reduce 步骤，主节点接收来自工作节点的结果并合并它们，以获得能够解决最初问题的总体结果和答案。

CouchDB 中的 Map/Reduce 特性生成键/值对，CouchDB 将它们插入到 B-树引擎中并根据它们的键进行排序。这就能通过键进行高效查找，并且提高 B-树中的操作的性能。此外，这还意味着可以在多个节点上对数据进行分区，而不需要单独查询每个节点。

传统的关系数据库管理系统有时使用锁来管理并发性，从而防止其他客户机访问某个客户机正在更新的数据。这就防止多个客户机同时更改相同的数据，但对于多个客户机同时使用一个系统的情况，数据库在确定哪个客户机应该接收锁并维护锁队列的次序时会遇到困难，这很常见。在 CouchDB 中没有锁机制，它使用的是多版本并发性控制（Multi version concurrency control，MVCC）— 它向每个客户机提供数据库的最新版本的快照。这意味着在提交事务之前，其他用户不能看到更改。许多现代数据库开始从锁机制前移到 MVCC，包括 Oracle（V7 之后）和 Microsoft® SQL Server 2005 及更新版本。

1.5 文档介绍

Couchdb 是一个用文档存储数据的数据库服务器。每个文档在数据库中都是唯一命名的，CouchDB 提供了一个 RESTful HTTP API，用于读取和更新（添加、编辑、删除）数据库文档。

文档是 CouchDB 中的主要数据单元，由任意数量的字段和附件组成。文档还包括由数据库系统维护的元数据。文档字段具有唯一的名称，并且包含各种类型的值（文本、数字、布尔值、列表等），并且没有对文本大小或元素计数的设置限制。

CouchDB 文档更新模型是无锁的、乐观的。文档编辑是由客户端应用程序

加载文档、应用更改并将其保存回数据库来完成的。如果编辑同一文档的另一个客户机先保存其更改，则客户机在保存时会收到一个编辑冲突错误。若要解决更新冲突，可以打开最新的文档版本，重新应用编辑，然后重试更新。

单个文档更新（添加、编辑、删除）要么全部成功，要么全部失败。数据库从不包含部分保存或编辑的文档。

1.6 ACID 介绍

couchdb 文件布局和提交系统具有所有原子一致的隔离持久性（acid）属性。在磁盘上，CouchDB 从不覆盖提交的数据或相关结构，确保数据库文件始终处于一致状态。这是一种“仅崩溃”的设计，在这种设计中，CouchDB 服务器不会经历关闭过程，它只能被终止。

文档更新（添加、编辑、删除）是序列化的，但同时写入的二进制 blob 除外。数据库读卡器从不被锁定，也不必等待编写器或其他读卡器。任何数量的客户机都可以在不被锁定或不被并发更新中断的情况下读取文档，即使是在同一文档上。CouchDB 读取操作使用多版本并发控制（MVCC）模型，其中每个客户机从读取操作的开始到结束都会看到数据库的一致快照。这意味着 CouchDB 可以保证每个文档的事务语义。

文档在 B 树中按其名称（docid）和序列 ID 进行索引。对数据库实例的每次更新都会生成一个新的序列号。序列 ID 稍后用于增量查找数据库中的更改。保存或删除文档时，这些 B 树索引会同时更新。索引更新总是发生在文件的末尾（仅追加更新）。

文档的优点是，数据已经方便地打包存储，而不是在大多数数据库系统中跨多个表和行进行拆分。当文档被提交到磁盘时，文档字段和元数据被打包到缓冲区中，依次排列一个文档和另一个文档（稍后有助于有效地构建视图）。

当 CouchDB 文档被更新时，所有数据和相关索引都被刷新到磁盘上，事务提交总是使数据库处于完全一致的状态。提交分两步进行：

所有文档数据和相关的索引更新都同步刷新到磁盘。

更新后的数据库头被写入两个连续的、相同的块中，以组成文件的第一个 4K，然后同步刷新到磁盘。

如果在步骤 1 中发生操作系统崩溃或电源故障，重新启动时只会忘记部分刷新的更新。如果在步骤 2（提交头）期间发生这种崩溃，那么将保留以前相同头的一个保留副本，以确保以前提交的所有数据的一致性。除了收割台区域，在发生故障或电源故障后，不需要进行一致性检查或修复。

1.7 压缩

浪费的空间通过偶尔的压缩来恢复。或者当数据库文件超过了一定的浪费空间时，压缩过程会将所有活动数据克隆到一个新文件中，然后丢弃旧文件。数据库始终保持完全联机，并且允许成功完成所有更新和读取。只有在复制了所有数据并且所有用户都转换到新文件时，才会删除旧的数据库文件。

1.8 视图

ACID 只处理存储和更新，但我们还需要能够以有趣和有用的方式显示数据。与必须小心地将数据分解为表的 SQL 数据库不同，CouchDB 中的数据存储在半结构化文档中。CouchDB 文档是灵活的，并且每个文档都有自己的隐式结构，这可以缓解双向复制表模式及其所包含数据的问题。

但是，除了充当一个奇特的文件服务器之外，一个用于数据存储和共享的简单文档模型太简单，无法在其上构建真正的应用程序——它只是做不到我们想要和期望的足够多的事情。我们希望以多种不同的方式对数据进行切片和切块。需要的是一种对尚未分解成表的数据进行筛选、组织和报告的方法。

1.8.1 视图模型

为了解决将结构添加回非结构化和半结构化数据的问题，CouchDB 集成了一个视图模型。 视图是聚合和报告数据库中文档的方法，可根据需要构建，以聚合，加入和报告数据库文档。 由于视图是动态构建的，并且不会影响基础文档，因此您可以根据需要使用相同数据的多个不同视图表示。

视图定义是严格虚拟的，只显示当前数据库实例中的文档，使它们与显示的数据分开并与复制兼容。 CouchDB 视图在特殊设计文档中定义，可以跨数据库实例（如常规文档）进行复制，因此不仅可以在 CouchDB 中复制数据，而且整个应用程序设计也可以复制。

1.8.2 视图索引

视图是数据库的实际文档内容的动态表示，而 CouchDB 可以轻松创建有用的数据视图。但是生成具有数十万或数百万个文档的数据库视图是耗费时间和资源的，并不是系统每次都应该从头开始做的事情。

为了保持视图快速查询，视图引擎维护其视图的索引，并逐步更新它们以反映数据库中的更改。 CouchDB 的核心设计主要针对高效，渐进式创建视图及其索引的需求进行了优化。

视图及其功能在特殊的“设计”文档中定义，设计文档可以包含任意数量的唯一命名视图函数。当用户打开视图并且其索引自动更新时，同一设计文档中的所有视图都将编入索引为单个组。

视图构建器使用数据库序列 ID 来确定视图组是否与数据库完全一致。如果不是，则视图引擎检查自上次刷新以来已更改的所有数据库文档（按打包顺序）。文档按照它们在磁盘文件中出现的顺序读取，从而降低了磁头搜索的频率和成本。

可以同时读取和查询视图，同时还可以刷新视图。如果客户端正在慢慢地流出大视图的内容，则可以在不阻塞第一个客户端的情况下为另一个客户端同时打开和刷新相同的视图。对于任何数量的同时客户端读者来说都是如此，他们可以读取和查询视图，同时为其他客户端刷新索引而不会给读者带来问题。

由于视图引擎通过“map”和“reduce”函数处理文档，因此它们之前的行值将从视图索引中删除（如果存在）。如果文档由视图函数选择，则函数结果将作为新行插入视图中。

当视图索引更改写入磁盘时，更新始终附加在文件末尾，用于减少磁盘提交期间的磁头搜索时间，并确保崩溃和电源故障不会导致索引损坏。如果在更新视图索引时发生崩溃，则不完整的索引更新将丢失并从其先前提交的状态逐步重建。

1.9 安全和验证

为了保护谁可以阅读和更新文档，CouchDB 有一个简单的读者访问和更新验证模型，可以扩展以实现自定义安全模型。

CouchDB 数据库实例具有管理员帐户。管理员帐户可以创建其他管理员帐户并更新设计文档。 设计文档是包含视图定义和其他特殊公式的特殊文档，以及常规字段和 blob。

当文档写入磁盘时，可以通过 JavaScript 函数动态验证它们的安全性和数据验证。 当文档通过所有公式验证标准时，允许更新继续。 如果验证失败，则更新将中止，并且用户客户端将收到错误响应。

用户凭证和更新文档都作为验证公式的输入，并可用于通过验证用户更新文档的权限来实现自定义安全模型。

基本的“仅作者”更新文档模型实现起来很简单，其中验证文档更新以检查用户是否列在现有文档的“作者”字段中。 还可以使用更多动态模型，例如检查单独的用户帐户配置文件以获取权限设置。

对实时使用和复制更新强制执行更新验证，确保在共享的分布式系统中进行安全性和数据验证。

1.10 分布式更新和复制

CouchDB 是一个基于对等的分布式数据库系统。它允许用户和服务器在断开

连接时访问和更新相同的共享数据。然后可以在以后双向复制这些更改。

CouchDB 文档存储，视图和安全模型旨在协同工作，使真正的双向复制高效可靠。文档和设计都可以复制，允许将完整的数据库应用程序（包括应用程序设计，逻辑和数据）复制到笔记本电脑以供脱机使用，或者复制到远程办公室中的服务器，在这些服务器中，缓慢或不可靠的连接使共享数据变得困难。

复制过程是增量的。在数据库级别，复制仅检查自上次复制以来更新的文档。如果由于网络问题或崩溃而导致复制在任何步骤失败，则下一次复制将在最后一个检查点重新启动。

可以创建和维护部分副本。可以通过 JavaScript 函数过滤复制，以便仅复制特定文档或满足特定条件的文档。这可以允许用户将大型共享数据库应用程序的子集脱机以供自己使用，同时保持与应用程序和该数据子集的正常交互。

1.10.1 冲突

冲突检测和管理是任何分布式编辑系统的关键问题。CouchDB 存储系统将编辑冲突视为一种常见状态，而非特殊状态。冲突处理模型简单且“非破坏性”，同时保留单个文档语义并允许分散的冲突解决。

CouchDB 允许在数据库中同时存在任意数量的冲突文档，每个数据库实例确定性地决定哪个文档是“赢家”，哪些是冲突。只有获胜文档才能显示在视图中，而“丢失”冲突仍然可以访问并保留在数据库中，直到数据库压缩期间被删除或清除。由于冲突文档仍然是常规文档，因此它们像常规文档一样进行复制，并遵循相同的安全性和验证规则。

当发生分布式编辑冲突时，每个数据库副本都会看到相同的获胜版本，并且每个副本都有机会解决冲突。解决冲突可以通过自动代理手动完成，也可以根据数据的性质和冲突完成。系统可以在保持单个文档数据库语义的同时实现分散式冲突解决。

即使多个断开连接的用户或代理尝试解决相同的冲突，冲突管理仍会继续有效。如果已解决的冲突导致更多冲突，则系统以相同的方式容纳它们，在每台机器上确定相同的获胜者并维护单个文档语义。

2 CouchDB 软件适配

2.1 安装 CouchDB 软件

2.1.1 从安装源安装

```
$ sudo apt-get install couchdb
```

从安装源安装后，deb 安装后脚本会自动对 couchDB 进行配置，并设置为服务启动，可直接进入后续验证步骤.

2.1.2 从源码安装

安装依赖：

```
$ sudo apt-get install build-essential pkg-config erlang \
libicu-dev libmozjs185-dev libcurl4-openssl-dev
```

编译：

```
$ ./configure
$ make release
```

执行 configure 时可以使用--help 参数查看帮助文档，添加需要的参数；如果执行 make 报错，可以尝试执行 gmake.

Make 成功时会显示如下信息：

```
... done
You can now copy the rel/couchdb directory anywhere on your system.
Start CouchDB with ./bin/couchdb from within that directory.
```

安装：

如 make 结束时的信息所示，将编译出的目录拷贝至想要安装的目录

创建 couchdb 用户：

```
adduser --system \
        --shell /bin/bash \
        --group --gecos \
        "CouchDB Administrator" couchdb
```

测试用户 shell 可用，并且拥有一个可写的主目录：

以 couchdb 用户登录；

运行 pwd 查看该用户的主目录.

将编译出的目录拷贝至 couchdb 用户的主目录（若上叙安装步骤中，已拷贝该目录到想安装的目录，则后续执行步骤可修改为该目录），并修改目录权限：

```
$ cp -R /path/to/couchdb/rel/couchdb /home/couchdb
$ chown -R couchdb:couchdb /home/couchdb
$ find /home/couchdb -type d -exec chmod 0770 {} \;
$ chmod 0644 /home/couchdb/etc/*
```

使用 couchdb 用户运行数据库：

```
$ sudo -i -u couchdb /home/couchdb/bin/couchdb
```

第一次运行可能会报错：

```
{database_does_not_exist,[{mem3_shards,load_shards_from_db,"_users" ...
```

这是提示我们要完成数据库的初始化工作，打开浏览器，输入地址：

```
http://127.0.0.1:5984/_utils#setup
```

选择初始化为单节点还是集群，完成初始化工作.

2.2 验证安装

```
$ curl http://127.0.0.1:5984
```

正确安装时，程序会返回类似以下的信息：

```
{"couchdb":"Welcome","uuid":"2a7b0688d64ca22abaafd1fbac75a3f7","version":"1.6.0","vendor":{"name":"Ubuntu","version":"16.04"}}
```

2.3 获取所有数据库

```
$ curl -X GET http://127.0.0.1:5984/_all_dbs
["_replicator","_users"]
```

2.4 创建数据库

```
$ curl -X PUT http://127.0.0.1:5984/baseball
{"ok":true}
$ curl -X GET http://127.0.0.1:5984/_all_dbs
["_replicator","_users","baseball"]
$ curl -X PUT http://127.0.0.1:5984/baseball
{"error":"file_exists","reason":"The database could not be created, the file already exists."}
```

2.5 删除数据库

```
$ curl -X PUT http://127.0.0.1:5984/plankton
{"ok":true}
$ curl -X DELETE http://127.0.0.1:5984/plankton
{"ok":true}
$ curl -X GET http://127.0.0.1:5984/_all_dbs
["_replicator","_users","baseball"]
```

2.6 Fauxton

Fauxton 是 CouchDB 提供的一套内置的 Web 管理界面，它实现了对 CouchDB 所有功能的完全访问，使用 Fauxton，我们可以创建和删除数据库，查看和编辑文档，撰写和运行 MapReduce 视图，以及复制数据库等.

在浏览器中输入以下内容访问 Fauxton 界面：

```
http://127.0.0.1:5984/_utils/
```

2.7 创建数据库和文档

在 Fauxton 中创建数据库很简单。在“概述”页面中，单击“创建数据库”。当要求输入名称时，输入 hello-world 并单击“创建”按钮。

创建数据库后，Fauxton 将显示其所有文档的列表。这个列表刚开始是空的，让我们创建第一个文档。单击“所有文档”旁边的加号，然后选择“新建文档”链接。CouchDB 将为您生成一个 UUID。

Fauxton 将显示新创建的文档及其_id 字段。要创建一个新字段，只需使用编辑器编写有效的 JSON。添加一个新字段，方法是在_id 值后面加一个逗号，然后添加文本：

```
"hello": "my new value"
```

单击绿色的“创建文档”按钮以完成创建文档。

文档的_rev 会自动重新生成。我们可以将_rev 认为是一个安全特性，只有和 CouchDB 文档的最新_rev 达成一致时，才可以成功保存更改。

2.8 MapReduce

传统的关系数据库允许运行任何数据结构正确的查询。而 CouchDB 使用预定义的 map 和 reduce 函数，称为 MapReduce。这些函数提供了很大的灵活性，因为它们可以适应文档结构的变化，并且可以独立和并行地计算每个文档的索引。在 CouchDB 术语中，一个 map 和一个 reduce 的组合称为一个视图。

对于经验丰富的关系数据库程序员，MapReduce 需要一些适应。reduce 查询不是声明要包含在结果集中的表的行，而是基于对映射函数生成的索引的简单范围请求，而取决于数据库来确定运行查询的最有效方式。

映射函数以每个文档为参数调用一次。函数可以选择全部跳过文档，或者将一个或多个视图行作为键/值对发出。映射函数不能依赖于文档之外的任何信息。这种独立性允许以增量和并行方式生成 CouchDB 视图。

CouchDB 视图存储为按键排序的行。这使得从一系列键中检索数据变得高效，即使在有数千或数百万行的情况下也是如此。在编写 CouchDB map 函数时，您的主要目标是构建一个索引，在附近的键下存储相关数据。

在运行示例 MapReduce 视图之前，我们需要先创建一些数据。我们将创建包含不同商店中各种超市商品价格的文档。让我们为苹果、桔子和香蕉创建文档。（可以使用 CouchDB 自动生成的_id 和_rev 字段。）使用 fauxton 创建具有如下最终 JSON 结构的文档：

```
{
    "_id": "00a271787f89c0ef2e10e88a0c0001f4",
    "_rev": "1-2628a75ac8c3abfffc8f6e30c9949fd6",
    "item": "apple",
    "prices": {
        "Fresh Mart": 1.59,
        "Price Max": 5.99,
```

```
        "Apples Express": 0.79
    }
}
```

```
{
    "_id": "00a271787f89c0ef2e10e88a0c0003f0",
    "_rev": "1-e9680c5d9a688b4ff8dd68549e8e072c",
    "item": "orange",
    "prices": {
        "Fresh Mart": 1.99,
        "Price Max": 3.19,
        "Citrus Circus": 1.09
    }
}
```

```
{
    "_id": "00a271787f89c0ef2e10e88a0c00048b",
    "_rev": "1-60e25d93dc12884676d037400a6fa189",
    "item": "banana",
    "prices": {
        "Fresh Mart": 1.99,
        "Price Max": 0.79,
        "Banana Montana": 4.22
    }
}
```

想象一下，我们正在为一个大型的午餐会做准备，但客户对价格非常敏感。为了找到最低价，我们将创建第一个视图，显示按价格排序的每个水果。单击“所有文档”返回 Hello World 概述，然后从“所有文档”加号中单击“新建视图”创建新视图。

命名设计文档_design/mydesigndoc，并将索引名称设置为 price。

编辑右侧的 map 函数，使其如下所示：

```
function(doc) {
    var shop, price, value;
    if (doc.item && doc.prices) {
        for (shop in doc.prices) {
            price = doc.prices[shop];
            value = [doc.item, shop];
            emit(price, value);
        }
    }
}
```

这是一个 javascript 函数，CouchDB 在计算视图时为每个文档运行。我们暂

时将 reduce 函数留空。

单击“保存文档，然后构建索引”，将看到按价格排序的结果行。如果按类型对项目进行分组，则会得到所有香蕉的价格排列在一起的结果。CouchDB 的关键字排序系统允许任何有效的 JSON 对象作为密钥。在本例中，我们将 emit 一个[item，price]数组，以便 couchdb 按 item 类型和价格分组。

修改视图功能（单击视图>左侧价格设计文档旁边的扳手图标，然后选择编辑），使其看起来像这样：

```
function(doc) {
    var shop, price, key;
    if (doc.item && doc.prices) {
        for (shop in doc.prices) {
            price = doc.prices[shop];
            key = [doc.item, price];
            emit(key, shop);
        }
    }
}
```

首先检查文档是否具有对我们有用的字段。CouchDB 可以从一些 map 函数故障中优雅地恢复，但是当一个映射函数定期失败（由于缺少必需字段或其他 javascript 异常）时，CouchDB 会关闭其索引以防止进一步使用。因此，在使用任何字段之前，检查它们是否存在是很重要的。在这种情况下，我们的 map 函数将跳过我们创建的第一个“hello world”文档，而不会 emit 任何行或遇到任何错误，直接显示此查询的结果。

一旦我们知道我们获取到一个带有项目类型和一些价格的文档，我们就迭代项目的价格并发出键/值对。键是项目和价格的数组，它构成了 CouchDB 排序索引的基础。在这种情况下，值是可以在其中找到所列价格项目的商店的名称。

视图行按其键排序——在本例中，首先按项目排序，然后按价格排序。这种复杂排序方法是使用 CouchDB 创建有用索引的核心。

MapReduce 对于已经花了多年时间使用关系数据库的开发人员具有一定的挑战性。需要记住的重要事项是，映射函数可以让您有机会使用所选的任何键对数据进行排序，而 CouchDB 的设计重点在于提供对键范围内数据的快速、高效访问。