

Projeto de Bloco C (Velocidade)

CouchDB

Componentes do Grupo:

Rafael Bahia Leandro

Vagner Praia da Silva

Íris Deberges

Wagner Rodrigues Alves

Índice

1. Divisão de assuntos	3
2. Introdução	4
3. Aplicação	4
4. Arquitetura	11
5. Instalação	17
6. Interação	19
7. Tarefas Administrativas	21
8. Conclusões	34
9. Referências Bibliográficas	34

1. Divisão de assuntos

Segue a especificação das seções e seus respectivos autores:

	Seção	Autor
A	Introdução, aplicações e organização de bibliografia	Wagner Rodrigues
B	Arquitetura, interação e conclusões	Rafael Bahia
C	Instalação e revisão/organização final de evidencias	Vagner Praia
D	Tarefas administrativas	Iris Deberges

2. Introdução

O que é o CouchDB?

Apache CouchDB, comumente referido como CouchDB, é um banco de dados de código-aberto que foca na facilidade de uso e na filosofia de ser "um banco de dados que abrange a Web". É um banco de dados não-relacional (NoSQL) que usa JSON para armazenar os dados, JavaScript como sua linguagem de consulta usando o MapReduce, e HTTP como API. Uma de suas características marcantes é a facilidade na replicação.

Ao contrário de um banco de dados relacional, o CouchDB não armazena os dados e relacionamentos em tabelas. Cada banco de dados é uma coleção de documentos independentes, e cada documento mantém seus próprios dados e esquemas. Uma aplicação pode acessar vários bancos de dados, por exemplo, no smartphone do usuário e outro em um servidor. Os metadados do documento contém informações de revisão, possibilitando mesclar quaisquer diferenças que possam ter ocorrido enquanto os bancos de dados estavam desconectados.

3. Aplicação

CouchDB é útil para muitas áreas de aplicação. Devido as suas características especialmente bem adequado para interações online através de documentos e gerenciamento de dados. Estes são o tipo de cargas de trabalho experimentadas pela maioria das aplicações web. Isso combinado com interface HTTP do CouchDB torná-lo um ajuste natural para a web.

Não há uma resposta certa sobre qual framework de desenvolvimento de aplicações deve ser usado com CouchDB. Já vimos aplicações de sucesso em quase todas as linguagens comumente usadas e estrutura.

Uma maneira diferente de modelar seus dados

CouchDB combina um modelo de armazenamento de documentos intuitivo com um poderoso mecanismo de consulta de uma forma que é tão simples que você provavelmente vai ser tentado a perguntar: "Por que ninguém construiu algo parecido com isso antes?"

O projeto de CouchDB empresta muito de arquitetura web e os conceitos de recursos, métodos e representações. Ele aumenta isso com poderosas formas de consulta, mapear, combinar e filtrar os dados. Adicionar a tolerância a falhas, escalabilidade extrema, e replicação incremental, e CouchDB define um ponto doce para bancos de dados de documentos.

Um melhor ajuste para Aplicações Comuns

Nós escrevemos software para melhorar as nossas vidas e as vidas dos outros. Geralmente isso envolve tomar algumas informações banais, como contatos, faturas ou recibos e manipulá-lo usando um aplicativo de computador. O CouchDB é um grande ajuste para aplicações comuns, como isso, porque abraça a ideia natural de documentos em evolução, auto-suficientes, como o próprio núcleo do seu modelo de dados.

Dados auto-suficiente

Uma fatura contém todas as informações pertinentes sobre uma única transação do vendedor, o comprador, a data, e uma lista de itens ou serviços vendidos. Como mostrado na Figura 1. Os documentos auto-contidos, não há nenhuma referência abstrata neste pedaço de papel que aponta para algum outro pedaço de papel com o nome e endereço do vendedor. Contadores apreciam a simplicidade de ter tudo em um só lugar. E dada a escolha, os programadores apreciam isso, também.

Real-world data is managed as real-world documents

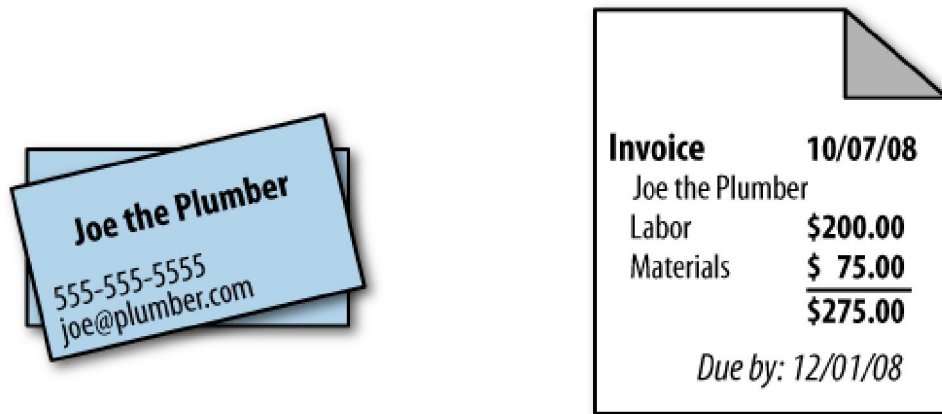


Figura 1. Os documentos auto contido

No entanto, usando referências é exatamente como modelar nossos dados em um banco de dados relacional! Cada fatura é armazenado em uma tabela como uma linha que se refere a outras linhas em outras tabelas uma linha para informações do vendedor, uma para o comprador, uma linha para cada item faturado, e mais linhas fixas para descrever os detalhes do item, detalhes do fabricante e assim por diante e assim por diante.

Este não é entendida como um distração do modelo relacional, que é amplamente aplicável e extremamente útil para uma série de razões. Felizmente, porém, ilustra o ponto que às vezes o seu modelo não pode "encaixar" os seus dados na forma como ocorre no mundo real.

Vamos dar uma olhada no banco de dados de contato humilde para ilustrar uma forma diferente de modelagem de dados, que mais de perto "encaixa" o seu homólogo do mundo real uma pilha de cartões de visita. Muito parecido com o nosso exemplo da fatura, um cartão de visita contém todas as informações importantes, bem ali na cartolina. Chamamos esses dados "auto suficientes" é um conceito importante para compreender as bases de dados de documentos como CouchDB.

Sintaxe e semântica

A maioria dos cartões de visita contém aproximadamente a mesma informação a identidade de alguém, uma afiliação, e algumas informações de contato. Enquanto a forma exata desta informação pode variar entre cartões de visita, as informações gerais a ser transportado permanece o mesmo, e nós somos facilmente capazes de reconhecê-lo como um cartão de visita. Neste sentido, podemos descrever um cartão de visita como um documento do mundo real .

Cartão de visita de Jan pode conter um número de telefone, mas nenhum número de fax, enquanto cartão de visita da J. Chris contém um telefone e um número de fax. O Jan não tem que fazer a sua falta de uma máquina de fax explícita por escrito algo tão ridículo como "Fax: Nenhum" no cartão de visita. Em vez disso, simplesmente omitir um número de fax implica que ele não tem um.

Podemos ver que os documentos reais do mesmo tipo, tais como cartões de visita, tendem a ser muito semelhantes em s emântica o tipo de informação que eles carregam, mas pode variar enormemente na s intaxe , ou como essa informação é estruturada. Como seres humanos, nós somos naturalmente confortável lidar com este tipo de variação.

Enquanto um banco de dados relacional tradicional requer que você modelar seus dados n a frente , design livre de esquema do CouchDB alivialo com uma poderosa forma de agregar seus dados a pós o fato , assim como fazemos com os documentos do mundo real. Nós vamos olhar em profundidade como projetar aplicativos com esse paradigma de armazenamento subjacente.

Blocos de construção para sistemas maiores

CouchDB é um sistema de armazenamento útil por si só. Você pode construir muitas aplicações com as ferramentas CouchDB lhe dá. Mas CouchDB é projetado com uma imagem maior em mente. Seus componentes podem ser usados como blocos de construção que resolvem problemas de armazenamento de formas ligeiramente diferentes para sistemas maiores e mais complexos.

Se você precisa de um sistema que é louco rápido, mas não é muito preocupado com a confiabilidade (pense em log), ou um que garante o armazenamento em dois ou mais locais separados fisicamente para a confiabilidade, mas você está disposto a assumir um acerto desempenho, CouchDB permite construir esses sistemas.

Há uma infinidade de botões que você poderia girar para fazer um sistema funcionar melhor em uma área, mas você vai afetar outra área quando fazê-lo. Um exemplo seria o teorema CAP discutido na consistência eventual . Para lhe dar uma ideia de outras coisas que afetam sistemas de armazenamento, ver Figura 2 e Figura 3 .

Ao reduzir a latência para um determinado sistema (e isso é verdade não só para sistemas de armazenamento), está a afetar a capacidade e rendimento.

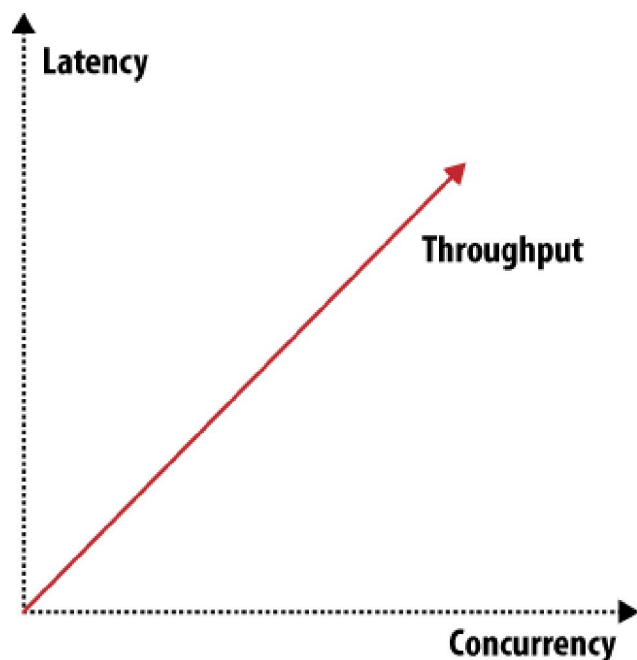


Figura 2. O throughput, latência, ou simultaneidade

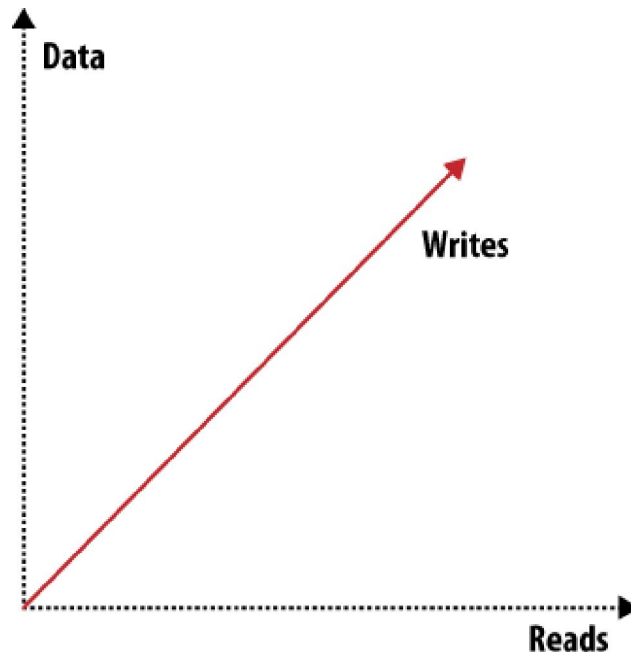


Figura 3. Escala: leia pedidos, solicitações de gravação ou de dados

Quando você quiser escalar, há três questões distintas para lidar com: Ciclismo pedidos ler, escrever solicitações e dados. Ortogonal a todas as três e os itens mostrados na Figura 2 e Figura 3 são muito mais atributos como a fiabilidade ou a simplicidade. Você pode desenhar muitos destes gráficos que mostram como diferentes características ou atributos de puxar em direções diferentes e, assim, moldar o sistema que eles descrevem.

CouchDB é muito flexível e dá-lhe blocos de construção suficientes para criar um sistema moldado para se adequar ao seu problema exato. Isso não é dizer que o CouchDB pode ser dobrado para resolver qualquer problema - CouchDB é nenhuma bala de prata -, mas na área de armazenamento de dados, você pode obter um longo caminho.

Dados locais é rei

CouchDB leva algumas lições aprendidas a partir da Web, mas há uma coisa que poderia ser melhorada sobre a Web: latência. Sempre que você tem que esperar por um pedido de resposta ou um site para renderizar, você quase sempre esperar por uma conexão de rede que não é tão rápido quanto você quer que ele nesse ponto. Esperar alguns segundos em vez de milissegundos afeta grandemente a experiência do usuário e, assim, a satisfação do usuário.

O que você faz quando você estiver offline? Isso acontece o tempo todo - o modem DSL ou cabo fornecedor tem problemas, ou o seu iPhone, Android ou Blackberry não tem nenhuma conectividade significa então há maneira de obter os seus dados.

CouchDB pode resolver este cenário, bem como, e isto é onde escala é importante novamente.

Desta vez, é a escala para baixo. Imagine o CouchDB instalado em telefones e outros dispositivos móveis que podem sincronizar dados com CouchDBs hospedados centralmente quando eles estão em uma rede. A sincronização não está vinculado por restrições de interface de usuário, como o tempo de resposta em segundos. É mais fácil de ajustar em alta largura de banda e maior latência do que para baixa largura de banda e latência muito baixa. As aplicações móveis podem então usar o CouchDB local para buscar dados, e uma vez que nenhuma rede remota é necessário para que a latência é baixa por padrão. Você pode realmente usar CouchDB em um telefone? Erlang, linguagem de implementação do CouchDB foi projetado para rodar em dispositivos embarcados magnitudes menores e menos poderosos do que os telefones de hoje.

Há uma abundância de exemplos de alguns sites e aplicativos que usam uma arquitetura CouchDB independente:

Damien Katz, inventor do CouchDB decidiu ver quanto tempo que seria necessário para implementar um calendário compartilhado com atualizações em tempo real como os eventos são alteradas no servidor. Demorou cerca de uma da tarde, graças a alguns incrível open source jQuery plug-ins.

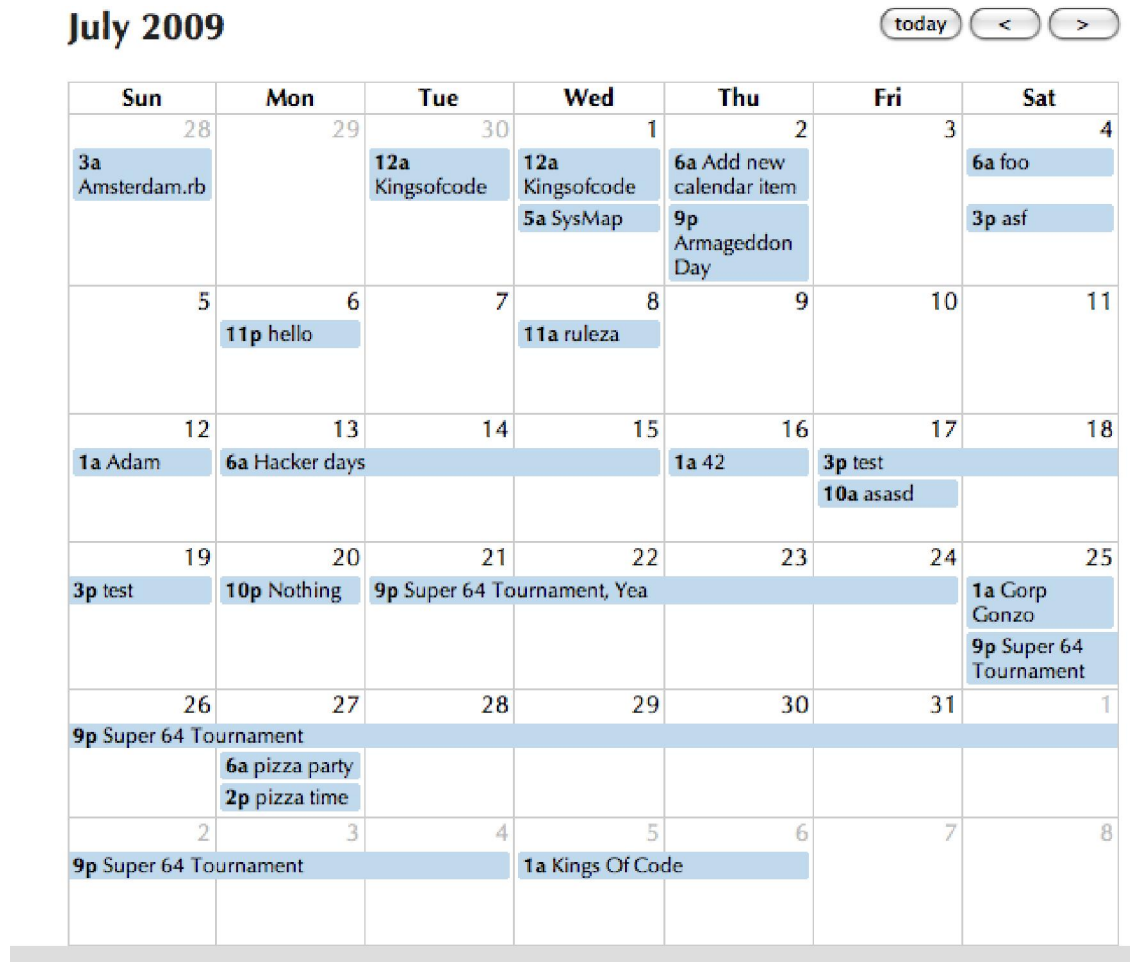



Figura 3. calendário de Grupo

Jason Davies trocou o backend do website Ely Serviço com o CouchDB, sem mudar nada visível para o usuário. Veja Figura 4, "Serviço de Ely" .

[Home](#) [Classic Services](#) ['Tin Boxes'](#) [A Message from the Owner](#) [History](#) [Contact Us](#) [Tel: 01353 662981](#)




Ely Service

Specialists In Classic Rovers

We specialise in classic Rovers (P4, P5 and P6 models) as well as other classic cars, ranging from a 1937 Austin 10 Cambridge through the decades to modern-day classics from the '90s.

Classic Car Services


- Basic servicing
- M.O.T.
- complete restoration, including:
- mechanical work
- bodywork
- paintwork
- interior trim
- rechroming
- supply and fitting of correct tyres for your classic.



[Read more...](#)

Modern Car Services


For those of you unlucky enough ever to have anything but a modern 'tin box'—but we love them really!—we offer you a similar level of service to that which our classic-car customers enjoy.



[Read more...](#)

History

The business first opened sometime in 1930, but the important date is March 1st, 1931, when Mr Charles Hull purchased the business. The following years saw the Company go from strength to strength, enjoying a



A Message from the Owner

I bought my first P6 in 1990 and they have gradually taken over. Over time I then progressed into P5 ownership as well. So when my chance came to take over at Ely, it seemed like manna from heaven!




Figura 4. Serviço Ely

Jason também converteu site de comércio eletrônico de sua mãe, Bet Ha Bracha, a um CouchApp. Veja Figura 5, "Bet Ha Bracha".

Processamento JS é um kit de ferramentas para a criação de arte de animação que é executado no navegador. Processamento JS Estúdio é uma galeria para esboços de processamento JS. Veja Figura 6, "Processamento de JS Studio".

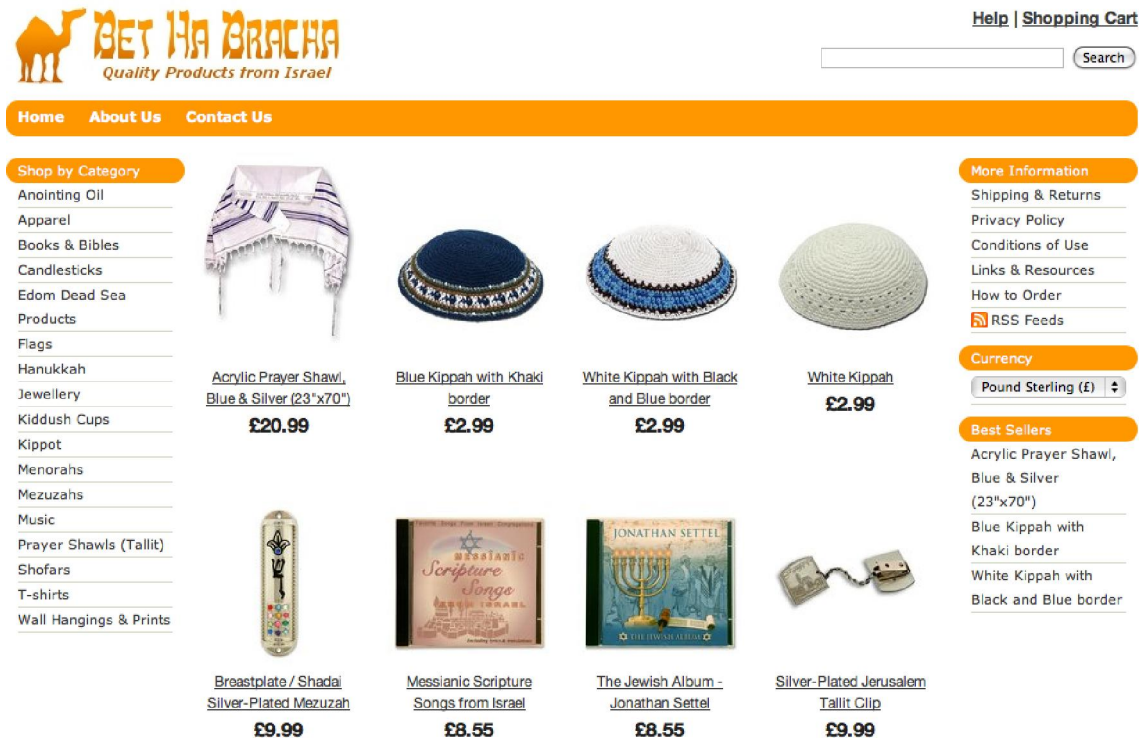


Figura 5. Bet Ha Bracha

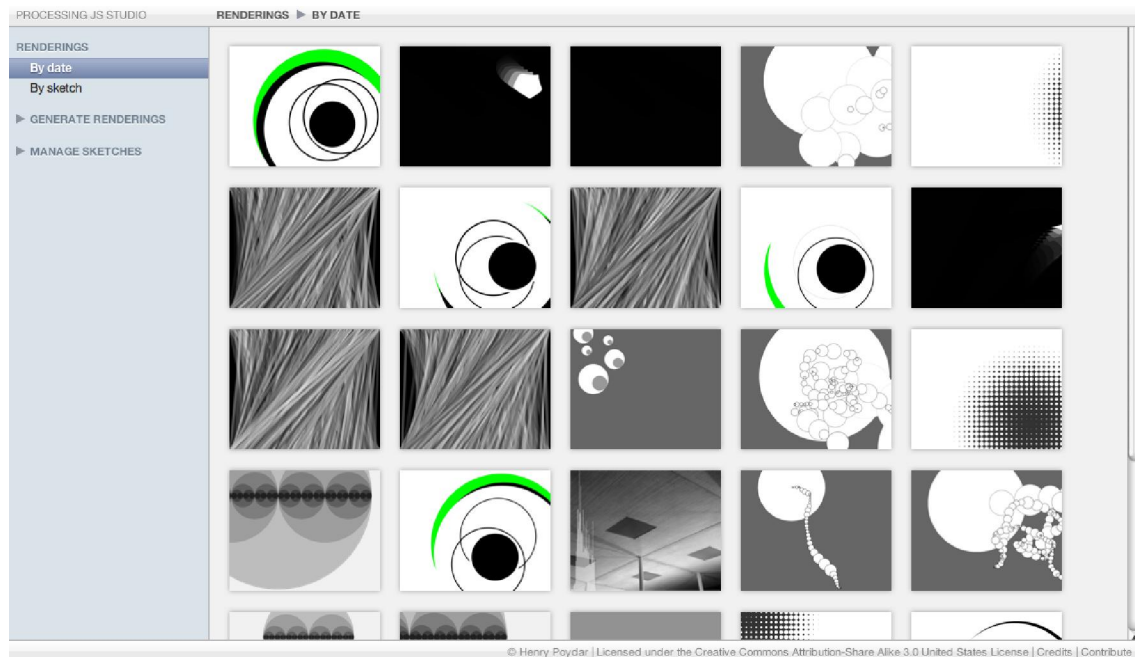


Figura 6. Processamento JS Estúdio

O Cliente CouchDB Twitter foi um dos primeiros CouchApps independentes para ser liberado. Ele está documentado no post do J. Chris, "Meu sofá ou seu, compartilhável Apps são o futuro" . A captura de tela na Figura 10, "Twitter Cliente" mostra a palavra nuvem gerada a partir

de uma visão MapReduce de tweets arquivados do CouchDB. A nuvem é normalizada contra a visão global, tão universalmente palavras não dominam o gráfico.



Figura 10. Twitter Cliente

4. Arquitetura

O couchDB está basicamente sustentado em um tripé que consiste em três componentes, a saber, JavaScript para consultas e scripting, HTTP para API e protocolo de comunicação e JSON para armazenamento. Nesta seção, falaremos resumidamente sobre cada um desses componentes na ordem apresentada, abordando também o MVCC (Multi Version Concurrency Control), mecanismo usado dentro do CouchDB para controle de alta concorrência no acesso a dados.

O couchDB utiliza o paradigma MapReduce e funções do Javascript para criação de índices e geração de Views de acesso aos dados. O JavaScript também pode ser utilizado para validação de documentos via funções similares aquelas usadas para o MapReduce. Cada vez que alguém tentar modificar um documento, o CouchDB passará para a função de validação uma cópia do documento existente, uma cópia do novo documento e um conjunto de informações adicionais, como detalhes de autenticação, por exemplo. A função de validação do JavaScript tem, assim, a oportunidade de aprovar ou negar a atualização. Essa validação em forma de função dentro do banco de dados e escrita em uma linguagem de programação conhecida e robusta faz com que uma enorme quantidade de ciclos de CPU sejam poupadas, ciclos estes que seriam desperdiçados serializando grafos de objetos do SQL, convertendo-os em objetos de domínio, e usando esses objetos para fazer a validação no lado da aplicação.

Realizar consultas no CouchDB é uma operação de duas fases: primeiro se cria um índice e depois se faz requisições para o índice. Isso é feito via JavaScript. Digamos que se quer uma lista de todos os documentos de nosso banco de dados ordenados por “last name”, o equivalente a “SELECT lastname FROM people ORDER BY lastname” in SQL. Assuma que temos os seguintes documentos já no banco de dados:

```
{
  "_id": "fd3ca61a",
  "name": {"first": "Beyoncé",
  "last": "Knowles"},
  "birthday": "1981-09-04"
}
{
  "_id": "2c87bab4",
  "name": {"first": "Dave",
  "last": "Grohl"},
  "birthday": "1969-01-14"
}
{
  "_id": "0ce27d5f",
  "name": {"first": "Henry",
  "last": "Rollins"},
  "birthday": "1961-02-13"
}
```

A definição (código) das Views do CouchDB são especificadas em documentos especiais chamados “documentos de design”. A única coisa que os faz diferentes dos demais documentos é que o ID (identificador único do documento) começa com `_design/`. Para criar um índice que é ordenado pelo último nome, pode-se escrever a seguinte função JavaScript:

```
function(doc) {
  emit(doc.name.last);
}
```

Esta função é chamada de “map function” pelo fato de ser executada durante a fase de Map do MapReduce. É necessário armazená-la dentro de um documento de design chamado, por exemplo, de `people/by_last_name`. Agora que tudo está no seu devido lugar, pode-se fazer a consulta:

curl http://127.0.0.1:5984/database/_design/people/view/by_last_name

Resultado:

```
{ "offset": 0, "rows": [
  { "key": "Grohl", "value": null, "id": "2c87bab4" },
  { "key": "Knowles", "value": null, "id": "fd3ca61a" },
  { "key": "Rollins", "value": null, "id": "0ce27d5f" }
]}
```

O que acontece por debaixo dos panos quando dispparamos esse HTTP Request pela primeira vez para uma view depois de tê-la criado em um documento de design? O engine do CouchDB cria o índice para suportar a view assim que percebe que a mesma não tem um. Baseado na função map reduce feita, o engine irá criar um par de chave-valor para cada documento contido na view e irá armazená-los em um arquivo em disco que é separado do arquivo principal do banco de dados. Quando esse processo de build do índice termina, ele abre o arquivo do índice e o lê de cima para baixo e retorna o resultado como foi visto acima. Agora, para cada requisição subsequente nessa view, o engine pode simplesmente abrir o índice e ler o resultado para o client. Antes disso, porém, o engine checa, usando o controle de alterações do

banco de dados, se há alterações no banco de dados desde que o índice em questão foi consultado. Se este for o caso, o engine irá incorporar essas novas mudanças dentro do índice antes de retornar o resultado do novo índice para o client. Isso mostra que, na arquitetura de consulta do CouchDB, é usado um esquema “preguiçoso” de construção de índices, ao invés de atualizá-los quando o dado é inserido ou atualizado, como é feito em outros sistemas de bancos de dados (relacionais, inclusive). O benefício aqui é duplo: se tivermos muitos índices, isso não diminui a velocidade de inserções e atualizações no banco de dados e, em segundo lugar, fazer atualizações de índices em batches ao invés de fazê-las uma a uma é muito mais eficiente em termos de espaço, tempo e recursos computacionais. Contudo, em vista dessa arquitetura, é importante se pensar no seguinte cenário: se tivermos um banco com vários milhões de documentos já inseridos nele, a primeira consulta que fizermos em uma view recém-criada pode demorar bastante tempo para completar. Isso porque, como vimos, o índice será criado obrigatoriamente para esse primeiro request. É como se, na primeira consulta a uma view, estivéssemos dando o seguinte comando em SQL: “ALTER TABLE CREATE INDEX”.

Tendo falando um pouco sobre a estrutura de consultas do CouchDB, falaremos agora um pouco sobre o protocolo de comunicação e de interação com o banco que foi adotado para este banco de dados, a saber, o HTTP. O HTTP é o protocolo “end-user” mais largamente utilizado em existencia. É fácil de compreender, poderoso, suportado em 100% dos ambientes de programação e vem com sua própria frota de hardware e software configuráveis que se encarrega de tudo desde roteamento, monitoração, proxying, debugging, etc. Poucos são os softwares tão ubíquos como HTTP.

O jeito principal de se fazer qualquer coisa com CouchDB é via HTTP. Criar um banco de dados: faça uma requisição HTTP; criar dados: faça uma requisição HTTP; consultar os dados: faça uma requisição HTTP; configurar replicação: faça uma requisição; configurar o banco de dados: faça uma requisição HTTP e assim por diante. Um exemplo básico disso é o comando abaixo que cria um banco de dados a partir da sua linha de comando:

```
curl -X PUT http://127.0.0.1:5984/my\_database
```

Um grande exemplo do papel do HTTP como protocolo de comunicação e como linguagem de interação com o banco é a interface de administração gráfica do CouchDB, a saber, o Futon. Todos os comandos realizados via Futon não precisam ser nem mesmo interpretados ou transformados porque tanto a API como a comunicação com o banco é feita via HTTP/REST. Isso faz com que muitos ciclos de CPU sejam poupados e o fluxo todo de requisição/resposta seja mais fluido. Outro ponto onde o HTTP se destaca na arquitetura do CouchDB é quando se fala de Consistent Hashing, que consiste em um modo simples de garantir que você pode sempre encontrar os documentos que você salvou, ao mesmo tempo que balanceia o load de armazenamento através das partições de rede. Pelo fato de usar HTTP, o CouchDB pode particionar os documentos de acordo com a URL de requisição sem inspecionar o corpo da requisição em si. Isso garante vários benefícios em um ambiente configurado em shard, por exemplo, simplificando o processo de armazenar e achar, posteriormente, um mesmo dado.

Agora que sabemos que para falar com o CouchDB usamos o protocolo HTTP, precisamos saber como é a resposta que o client recebe. Tais respostas obedecem, por padrão, um modelo chamado JSON (JavaScript Object Notation). Esse mesmo modelo é o padrão do storage engine do CouchDB e, sendo um formato leve e baseado em um sub-conjunto da sintaxe do JavaScript, oferece grande capacidade de armazenamento e consulta.

Um dos melhores fatores relacionados ao JSON é que é fácil de ler e escrever “à mão”, muito melhor quando comparado a algo como XML, por exemplo. Podemos fazer “parse” dele naturalmente com JavaScript porque a mesma sintaxe é compartilhada. Isso diminui drasticamente o que é chamado de incompatibilidade de impedancia entre banco de dados e programação. Além disso, JSON contém também um sub-conjunto de todos os tipo de dados nativos compartilhados entre todos os ambiente de desenvolvimento, ou seja, números, booleans,

strings, lists e hashes, etc. Isto faz com que o JSON seja um excelente formato para troca de dados entre diferentes sistemas porque tudo o que você precisa é de um “parser” de JSON que traduza JSON em tipos nativos de uma determinada linguagem de programação. Adicionalmente, JSON foi feito pensando em programação voltada para a WEB, por ser extremamente conciso e possuir boa capacidade de compressão de dados, logo é uma escolha lógica para programação de aplicativos móveis. Com CouchDB, você já possui JSON “out of the box”!

CouchDB armazenará qualquer JSON que você mandar. Neste sentido, CouchDB é um banco de dados “schemaless” ou “schema free” por padrão. Esta abordagem facilita o desenvolvimento de aplicações iniciantes, à medida que o programador não precisa gastar horas definindo o modelo de dados antecipadamente. Você simplesmente começa a programar e serializar seus objetos em formato JSON armazenando-os no CouchDB. Isso também corta fora a camada intermediária conhecida como Object Relational Mappers (ORMs), que faz a “tradução” do que está armazenado no banco de dados relacional para um formato que seja mais compreensível para o desenvolvedor. Com o CouchDB, você tem a mesma interface de forma nativa, deixando uma série de problemas relacionadas a incompatibilidade de impedancia para trás desde o início de seu desenvolvimento.

Em seu modelo de armazenamento de dados com JSON, o CouchDB também suporta o tipo de dados binário. O mecanismo para tal é chamado de “attachments”, e funciona como anexos de email: o dado binário é armazenado sob um nome e com um tipo de conteúdo correspondente na parte do documento JSON denominado “_attachments”. Apesar de ser “schema free”, o CouchDB possui mecanismos que permitem controlar mais o schema dos documentos armazenados. Tudo o que você tem que fazer é escrever uma função JavaScript que decide se o documento está em conformidade com o padrão esperado ou não.

Ainda falando sobre armazenamento, é importante saber que cada banco de dados dentro do CouchDB é suportado por um simples arquivo no sistema de arquivos do sistema operacional. Tudo o que vai para o banco, vai para esse arquivo. Índices para views utilizam o mesmo mecanismo de armazenamento, mas cada view tem o seu próprio arquivo no sistema de arquivos que está separado do arquivo principal do banco de dados. Ambos os arquivos são operados em uma forma append-only e a estrutura de dados utilizada nos mesmos é a B-Tree ou árvore binária. Mesmo quando documentos são deletados, a informação vai para o final do arquivo. O resultado é uma extrema resiliência contra perda de dados. Uma vez que o dado foi comitado para o arquivo e o arquivo foi escrito para o disco, CouchDB nunca tentará sobrescrever, total ou parcialmente, aquele dado. Isso garante que em qualquer cenário de erro (queda software ou de hardware, queda de luz, disco cheio, etc.) CouchDB garante que os dados previamente comitados estão intactos. O único real problema que pode ocorrer é quando o disco ou sistema de arquivos corrompe o dado, e mesmo assim o CouchDB utiliza checksums para detectar esse tipo de erro.

Adicionalmente, essa forma de operação (append-only) no final do arquivo permite que a camada de storage opere em largos e convenientes batches sem muitos seeks. Este é o melhor cenário tanto para discos mecânicos como para SSDs modernos. Em contrapartida, o CouchDB gasta recursos fazendo o que é chamado de “compaction”, que é o processo de limpar antigas versões de documentos, liberando espaço no banco de dados e consequentemente em seu arquivo. Compaction percorre a lista de mudanças de um banco de dados desde o início e copia as mais recentes versões dos documentos para um novo arquivo. Feito isso, o arquivo antigo é trocado pelo novo e o antigo arquivo é deletado.

As versões de documentos mencionadas acima fazem parte do MVCC (Multi Version Concurrency Control), modelo através do qual o CouchDB faz o controle de concorrência no banco de dados. Ao invés de utilizar locks (mecanismo de controle de concorrência usado por padrão pela maioria dos bancos transacionais), CouchDb usa o MVCC para garantir que o banco de dados rodará em máxima velocidade em todo o tempo, mesmo debaixo de alto load. Requisições são processadas em paralelo, fazendo excelente uso de cada recurso de processamento que o seu servidor tem a oferecer.

Para tal, documentos no CouchDB são versionados, tal como seria versionado um arquivo que estivesse em um sistema como o Subversion. Se você quer mudar o valor em um documento, você cria uma nova versão daquele documento e salva no lugar do antigo. Depois de fazer isso, você terá duas versões do mesmo documento, um velho e um novo. Como isso é melhor do que locks? Considere um conjunto de requisições querendo acesso a um documento. A primeira requisição lê o documento. Enquanto isso está sendo processado, uma segunda requisição altera o documento. A segunda requisição inclui uma completamente nova versão do documento, e o CouchDB pode simplesmente anexá-la ao banco de dados sem ter que esperar pela requisição que está lendo o mesmo documento. Quando uma terceira requisição quiser ler o mesmo documento, o CouchDB apontará para a nova versão que acabou de ser escrita. Durante todo esse processo, a primeira requisição pode ainda estar lendo a versão original sem ser interrompida ou atrapalhada. Uma requisição de leitura sempre verá a versão mais recente de seu banco de dados. Uma requisição de escrita sempre tem que passar a versão do documento que deseja alterar. Abaixo segue uma imagem ilustrativa comparando MVCC e Locks.

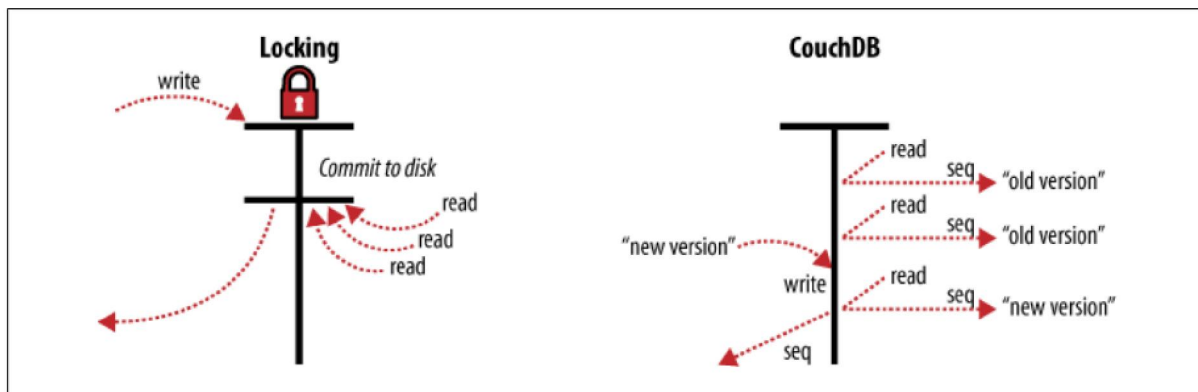


Figura 1 - MVCC Vs Locks

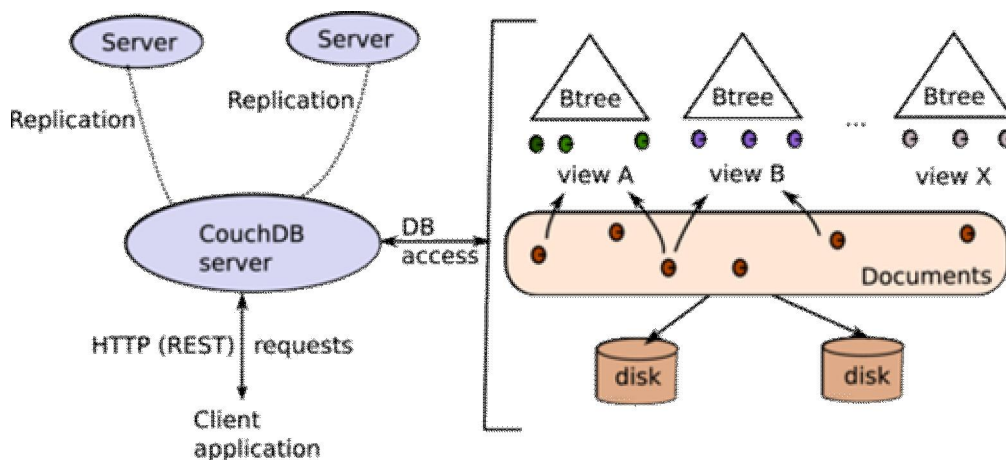


Figura 2 - Storage Engine

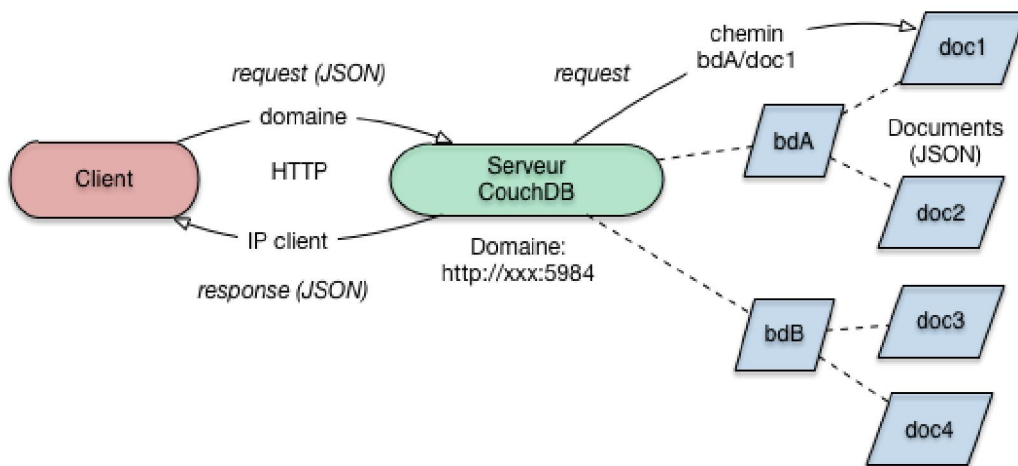


Figura 3 - Bancos e Documentos

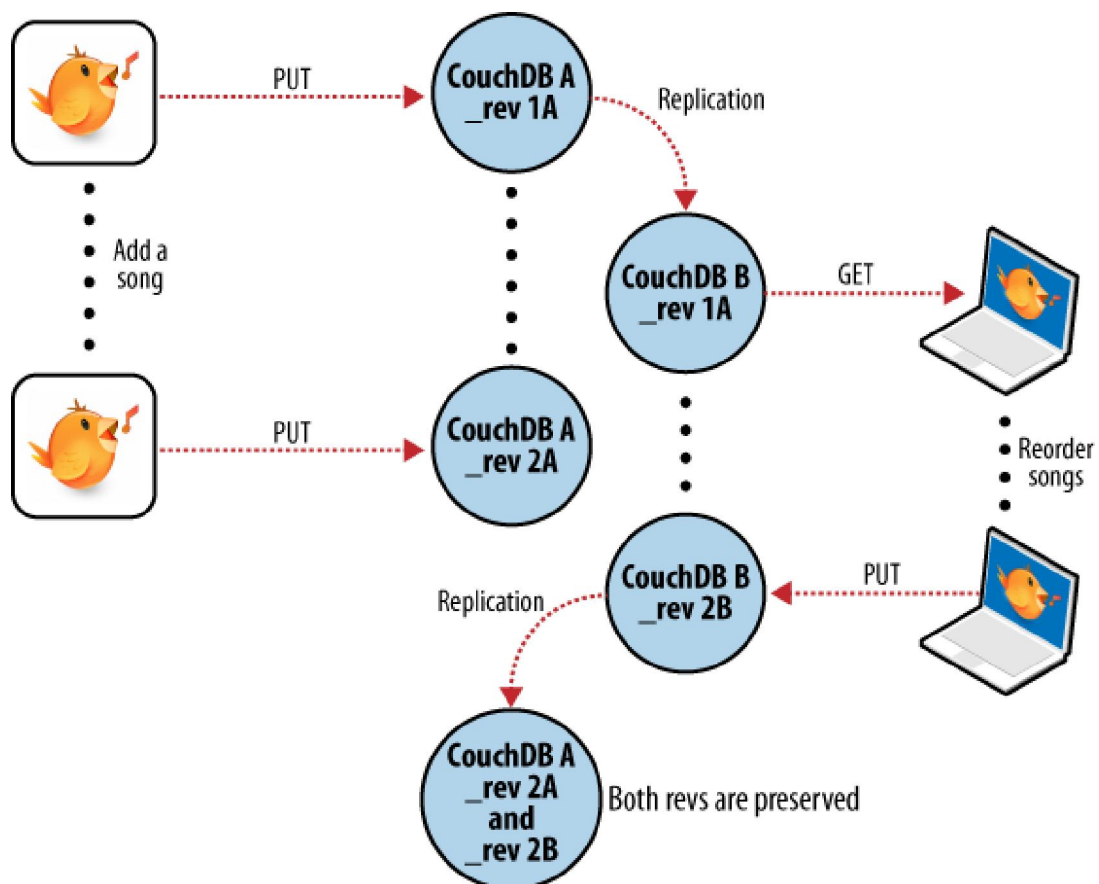


Figura 4 - Replicação e MVCC

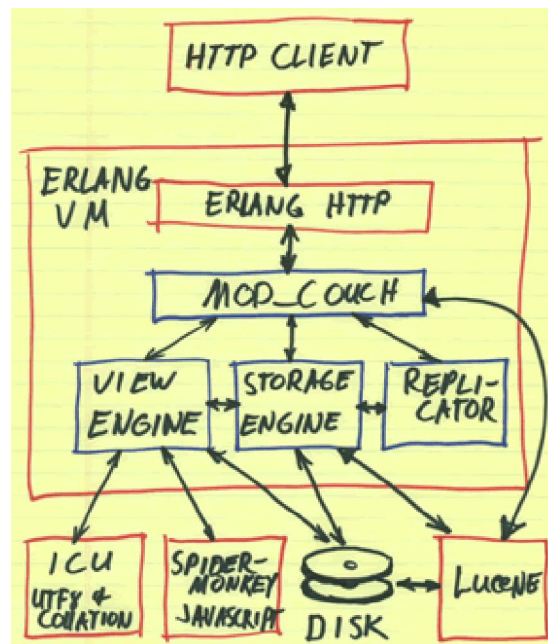


Figura 5 - Visão Geral

5. Instalação

O CouchDB está disponível ser instalado em praticamente todos os principais sistemas operacionais do mercado. Esta instalação pode ser realizada de forma bem prática e rápida, em praticamente qualquer computador, uma vez que não existem requisitos mínimos de hardware necessários para a instalação. Para a maioria dos casos, existem duas maneiras para se instalar o CouchDB:

- Por meio de um processo automatizado, como utilizando alguma suíte de pacotes das distribuições Linux ou utilizando um arquivo de instalação para Windows.
- Ou por meio do build do código fonte disponível no site do CouchDB.

A documentação disponível no site do CouchDB traz informações sobre como fazer a instalação nos sistemas operacionais Unix, Windows, Mac OS X e FreeBSD. A seguir vamos descrever a instalação dos sistemas operacionais mais utilizados, Linux, Windows e Mac OS.

Instalação no Linx

Em sistemas operacionais Linux, a instalação pode ser realizada utilizando a suíte de pacotes de software da plataforma escolhida. Por exemplo, no Debian a instalação pode ser realizada utilizando o aptitude com o seguinte comando:

```
sudo aptitude install couchdb
```

No Ubuntu a instalação pode ser feita pelo meio do APT, usando o seguinte comando:

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

Já no CentOS, o YUM pode ser utilizado com o comando:

```
yum install couchdb
```

Estas mesmas suítes de pacotes podem ser utilizadas para a instalação das dependências do CouchDB que são as seguintes:

- Erlang OTP (\geq R14B01, \leq R17)
- ICU
- OpenSSL
- Mozilla SpiderMonkey (1.8.5)
- GNU Make
- GNU Compiler Collection
- libcurl
- help2man
- Python (\geq 2.7) for docs
- Python Sphinx (\geq 1.1.3)

Instalação em Windows

A instalação no Windows pode ser realizado utilizando o arquivo de instalação disponível na página <http://couchdb.apache.org/#download>. Nesta forma de instalação é preciso somente seguir os passos solicitados durante a execução do arquivo.

A instalação ainda pode ser realizada por meio do build do código fonte utilizando o compilador do Cygwin. Nesta forma de instalação ainda tem as seguintes dependências:

- Erlang OTP (\geq 14B01, $<$ R17)
- ICU (\geq 4.*)
- OpenSSL ($>$ 0.9.8r)
- Mozilla SpiderMonkey ($=$ 1.8.5)
- Microsoft SDK 7.0 or 7.1
- libcurl (\geq 7.20)
- help2man
- Python (\geq 2.7) for docs
- Python Sphinx (\geq 1.1.3)

Instalação no Mac OS X

A instalação no Mac OS pode ser realizada pelo simples processo de baixar o arquivo Zip disponível na página <http://couchdb.apache.org/#download>, descompactá-lo e adicionar o arquivo CouchDB.app, presente dentro do Zip, na pasta de aplicações do sistema operacional.

Ainda há como fazer a esta instalação com o HomeBrew e MacPorts. Com o HomeBrew há a necessidade da instalação dos seguintes pacotes:

- autoconf
- autoconf-archive
- automake
- libtool
- erlang
- icu4c
- spidermonkey

- curl

Após estas instalações, o comando para instalar o CouchDB é o seguinte:

brew install couchdb

Com MacPorts a instalação é realizada com o seguinte comando:

port install couchdb

6. Interação

Como já foi mencionado, a linguagem de consulta utilizada no CouchDB é o Javascript. Através dela são montadas as funções Map e Reduce que dão origem as views que tem os documentos como referencia. Contudo, por estar baseado fortemente na arquitetura web (mais especificamente no HTTP), a forma de interação com os documentos em si não é o javascript, e sim, os métodos de request que são próprios do HTTP. Veja os exemplos a seguir para melhor entendimento.

- a. Trazendo a lista de bancos de dados existentes

```
C:\curl>curl -X GET http://127.0.0.1:5984/_all_dbs
["_replicator","_users","albums","baseball","hello-replication","hello-world","test_suite_db","test_suite_db2"]
```

Note a presença do método http “GET”, tão comum em páginas web. Este método também pode ser usado para capturar documentos dentro de um banco de dados, como ver-se-á mais adiante.

- b. Criando um novo banco de dados

```
C:\curl>curl -X PUT http://127.0.0.1:5984/infnet
{"ok":true}

C:\curl>curl -X GET http://127.0.0.1:5984/_all_dbs
["_replicator","_users","albums","hello-world","infnet","test_suite_db","test_suite_db2"]
```

- c. Dropando um banco de dados

```
C:\curl>curl -X DELETE http://127.0.0.1:5984/infnet
{"ok":true}

C:\curl>curl -X GET http://127.0.0.1:5984/_all_dbs
["_replicator","_users","albums","hello-world","test_suite_db","test_suite_db2"]
```

- d. Adicionando um novo documento banco de dados “albums”

```
C:\curl>curl -X PUT http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c28011e7 \ -d '{"title":"","There is Nothing Left to Lose","artist":"","Foo Fighters"}'
{"ok":true,"id":"54a16d264b6e7ea277feed06c28011e7","rev":"1-4b39c2871c9ad54cb37e08fa02fec636"}
```

Note que para inserir o documento é preciso passar a sua URL completa, como se estivesse acessando a um site. Nesta URL devem estar contidos o IP e porta em que o CouchDB está escutando as conexões, o banco de dados e o id do documento que o identificará. Para

conferir se o documento foi realmente inserido, basta usar o método GET passando a URL do documento, como é feito abaixo:

```
C:\curl>curl -X GET http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7
{"_id":"54a16d264b6e7ea277feed06c20011e7","_rev":"1-4b39c2971c9ad54cb37e08fa02fec636","title":"There is Nothing Left to Lose","artist":"Foo Fighters"}
```

e. Atualizando um documento

```
C:\WINDOWS\system32\cmd.exe
C:\curl>curl -X PUT http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7 -d '{"title":"There is Nothing Left to Lose","artist":"Foo Fighters"}'
{"ok":true,"id":"54a16d264b6e7ea277feed06c20011e7","rev":"4-664ac0dd5f86aeb5ef9c4a942251853a"}
C:\curl>curl -X PUT http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7 -d '{"_rev":"4-664ac0dd5f86aeb5ef9c4a942251853a","title":"There is Nothing Left to Lose","artist":"Foo Fighters"}'
{"ok":true,"id":"54a16d264b6e7ea277feed06c20011e7","rev":"5-c42abed7e63085cd6919bc77f40ae622"}
```

Aqui tem uma observação muito importante a se fazer. Apesar de o mesmo método PUT ser utilizado para inserir um documento, é importante saber que ao usá-lo para atualizar um documento, deve ser passada versão do documento que se quer atualizar. Isso porque o CouchDB utiliza MVCC (Multi version Concurrency Control) e armazena diversas versões de um mesmo documento, mesmo que ele tenha sido já apagado. Caso a versão do documento que se quer atualizar não for passada, tem-se o erro abaixo:

```
C:\curl>curl -X PUT http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7 -d '{"_rev":"","title":"There is Nothing Left to Lose","artist":"Foo Fighters","year":"1997"}'
{"ok":true,"id":"54a16d264b6e7ea277feed06c20011e7","rev":"5-c42abed7e63085cd6919bc77f40ae622"}
C:\curl>curl -X PUT http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7 -d '{"title":"There is Nothing Left to Lose","artist":"Foo Fighters","year":"1997"}'
{"error":"conflict","reason":"Document update conflict"}
```

A cada alteração, é gerada uma nova versão para o documento. Veja que, nesse caso, a versão do documento é 5 após o último PUT, quando era 4.

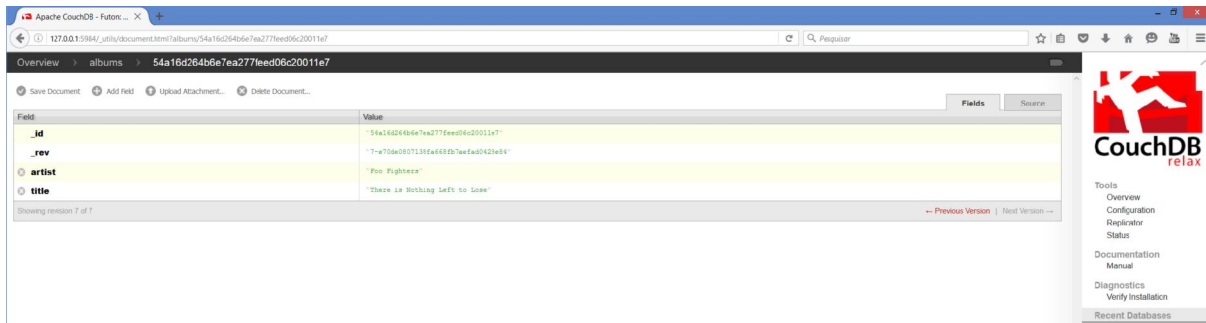
```
C:\curl>curl -X GET http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7
{"_id":"54a16d264b6e7ea277feed06c20011e7","_rev":"5-c42abed7e63085cd6919bc77f40ae622","title":"There is Nothing Left to Lose","artist":"Foo Fighters","year":"1997"}
```

f. Removendo um documento

Por fim, o método DELETE apaga o documento. Note que uma nova versão é gerada (versão número 6). Essa versão continuará no banco de dados até que o CouchDB resolva todas as pendências relacionadas a replicação e outros mecanismos relacionados a consistência eventual.

```
C:\curl>curl -X DELETE http://127.0.0.1:5984/albums/54a16d264b6e7ea277feed06c20011e7?rev=5-c42abed7e63085cd6919bc77f40ae622
{"ok":true,"id":"54a16d264b6e7ea277feed06c20011e7","rev":"6-662150627cbe94c4ba28e1ce754963a"}
```

É importante mencionar a interface gráfica de administração que já vem junto com o CouchDB. Essa interface chama-se Futon. Através dela é possível fazer todas as operações com documento demonstradas acima, além de várias operações administrativas relacionadas a configuração do banco, replicação, etc. Segue uma imagem da interface de administração. Perceba os comandos de manipulação de documento à esquerda e os comandos de configuração no menu lateral à direita.



7. Tarefas Administrativas

Segurança :

Mecanismos de segurança básicas em CouchDB. Começando pela administração, autenticação básica, autenticação Cookie, e OAuth.

Administração :

Quando você inicia, CouchDB, ele permite que qualquer solicitação seja feito por qualquer pessoa. Como Criar um banco de dados e excluir algum documento. Todos usuarios Começam como Administradores, ele parte desse Principio. O CouchDB chama isso de iniciando como administrador. Todo mundo tem privilégios para fazer qualquer coisa.

Embora seja incrivelmente fácil de começar com o CouchDB dessa forma, deveria ser óbvio que colocar uma instalação padrão pois dessa forma. Qualquer usuario pode excluir o banco de dados.

Por padrão, CouchDB vai escutar somente em sua interface de rede de auto-retorno (127.0.0.1 ou localhost) e, portanto, só você será capaz de fazer pedidos de CouchDB ou quem tiver acesso ao servidor fisico ou remoto. Mas quando você começa a abrir seu CouchDB ao público (isto é, dizendo-lhe para ligar para o endereço IP público da sua máquina), você vai querer pensar sobre a restrição de acesso para que nenhum usuario tenha acesso de administração do CouchDB .

CouchDB tem a idéia de um usuário administrador (por exemplo, um administrador, um super Usuario ou root) que é permitido fazer qualquer coisa em uma instalação do CouchDB. Por padrão, todo mundo é um administrador. Se você não gosta disso, você pode criar usuários admin específicos com um nome de usuário e senha como suas credenciais.

CouchDB também define um conjunto de solicitações que os usuários administradores têm permissão para fazer. Se você tiver definido um ou mais usuários específicos de administração, CouchDB vai pedir a identificação de certos pedidos:

Criação de um banco de dados (PUT / banco de dados)

A exclusão de um banco de dados (APAGAR / banco de dados)

Criando um documento de design (PUT / database / _design / app)

Atualizando um documento de design (PUT / database / _design / app? Rev = 1-4E2)

A exclusão de um documento de design (DELETE / database / _design / app? Rev = 1-6A7)

Provocando compactação (POST / _compact)

Lendo a lista de status da tarefa (GET / _active_tasks)
Reiniciar o servidor (POST / _restart)
Lendo a configuração ativa (GET / _config)
Atualizar a configuração ativa (PUT / _config)

Criando usuários :

Para diminuir os riscos, recomendamos aplicar as seguintes medidas para sua proteção. Pode se utilizar a API : "http://127.0.0.1:5984/_utils" que acessará o gerenciador web do CouchDB.

1 – Criar um usuário administrador no servidor clicando no botão “Fix this!”, localizado no canto inferior direito.

2 – Criar um usuário no-admin e atribuí-lo (por nome ou papel) para ser um usuário administrador do banco de dados em específico. Isso pode ser feito através do ícone “Segurança” no topo do gerenciador, quando você está em um banco de dados específico. Ou então criar este non-admin através do HTTP API.

3 – Criar um usuário non-admin no CouchDB e atribuí-los (por nome ou papel) para ser apenas leitor (read) no banco de dados em algum banco de dados específico. Isso pode ser feito através do ícone “Segurança” no topo do gerenciador quando você está em um banco de dados específico. Ou então criar este non-admin através do HTTP API.

4 – Criar um usuário non-admin no CouchDB e criar um documento de design de banco de dados que inclui uma função de validação, especificamente em uma propriedade “validate_doc_update” no documento de design. O valor dessa propriedade é uma função (que você escreve) para verificar um nome de usuário ou regra no argumento userCtx que é passado para a função específica, assim poderia alertar um erro na função se o usuário ou a regra não é quem pode escrever no banco de dados.

5 – Como medida adicional de proteção, o CouchDB disponibiliza a autenticação via Cookie, bastando enviar uma requisição para a API com o usuário e senha já presentes no mesmo. Por padrão, cada token tem sua duração de 10 minutos.

6 – Criando um usuário chamado zezinho.

```
# curl -X PUT http://192.168.0.64:5984/_users/org.couchdb.user:zezinho -d '{"name": "zezinho",  
"password": "S3nhaS3cr3t4", "roles": [], "type": "user"}'
```

7 – Listando usuários.

```
# curl http://192.168.0.64:5984/_users/_all_docs
```

8 – Excluindo um database.

```
# curl -X DELETE http://192.168.0.64:5984/new_db
```

Autenticação Básica

Autenticação básica (RFC 2617) é uma maneira rápida e simples para autenticar com o CouchDB. A principal desvantagem é a necessidade de enviar as credenciais do usuário em cada solicitação que pode ser inseguro e pode prejudicar o desempenho operação (dessa forma CouchDB deve calcular o hash de senha em cada pedido):

Request:

```
GET / HTTP/1.1
Accept: application/json
Authorization: Basic cm9vdDpyZWxheA==
Host: localhost:5984
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: must-revalidate
Content-Length: 177
Content-Type: application/json
Date: Mon, 03 Dec 2012 00:44:47 GMT
Server: CouchDB (Erlang/OTP)

{
  "couchdb": "welcome",
  "uuid": "0a959b9b8227188afc2ac26ccdf345a6",
  "version": "1.3.0",
  "vendor": {
    "version": "1.3.0",
    "name": "The Apache Software Foundation"
  }
}
```

Autenticação de Cookie

Para a autenticação de cookies (RFC 2109) CouchDB gera um sinal de que o cliente pode usar para as próximas solicitações para CouchDB. Tokens são válidos até um tempo limite. Quando CouchDB vê um token válido em um posterior pedido, ele irá autenticar o usuário por este token sem pedir a senha novamente. Por padrão, os cookies são válidos por 10 minutos, mas é ajustável. Também é possível fazer cookies persistentes.

Para obter o primeiro token e, assim, autenticar um usuário, pela primeira vez, o nome de usuário e senha devem ser enviadas para o API_SESSION.

/_sessão

POST /_SESSION

Inicia nova sessão para credenciais do usuário especificados, fornecendo valor Cookie.

Cabeçalhos de solicitação:

Tipo de conteúdo -

application / x-www-form-urlencoded

application / json

Parâmetros de Consulta:

next (string) - Enforces redireciona após o login bem-sucedido para o local especificado. Esta localização é relativo da raiz do servidor. Opcional.

Parâmetros forma:

nome - nome de usuário

password - Password

Cabeçalhos de resposta:

Set-Cookie - token de autorização

Resposta JSON objeto:

ok (boolean) - Status de Operação

nome (string) - Nome de usuário

papéis (array) - Lista de funções de usuário

Códigos de status:

200 OK - autenticado com êxito

302 Encontrado - redirecionamento após a autenticação bem sucedida

401 Unauthorized - Nome de usuário ou senha não foi reconhecido

Request:

```
POST /_session HTTP/1.1
Accept: application/json
Content-Length: 24
Content-Type: application/x-www-form-urlencoded
Host: localhost:5984

name=root&password=relax
```

It's also possible to send data as JSON:

```
POST /_session HTTP/1.1
Accept: application/json
Content-Length: 37
Content-Type: application/json
Host: localhost:5984

{
  "name": "root",
  "password": "relax"
}
```


Response:

```
HTTP/1.1 200 OK
Cache-Control: must-revalidate
Content-Length: 43
Content-Type: application/json
Date: Mon, 03 Dec 2012 01:23:14 GMT
Server: CouchDB (Erlang/OTP)
Set-Cookie: AuthSession=cm9vdDo1MEJCRkYwMjQ0L09yL0IwShrgt8y-UkhI-c6BGw; Version=1; Path=/; HttpOnly

{"ok":true,"name":"root","roles":["_admin"]}
```

Se no próximo parâmetro de consulta foi fornecido a resposta irá acionar o redirecionamento para o local especificado em caso de autenticação bem-sucedida:

Request:

```
POST /_session?next=/blog/_design/sofa/_rewrite/recent-posts HTTP/1.1
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Host: localhost:5984

name=root&password=relax
```

Response:

```
HTTP/1.1 302 Moved Temporarily
Cache-Control: must-revalidate
Content-Length: 43
Content-Type: application/json
Date: Mon, 03 Dec 2012 01:32:46 GMT
Location: http://localhost:5984/blog/_design/sofa/_rewrite/recent-posts
Server: CouchDB (Erlang/OTP)
Set-Cookie: AuthSession=cm9vdDo1MEJDMDEzRTp7Vu5GKCKTxTVxwXbpXsBARQWnhQ; Version=1; Path=/; HttpOnly

{"ok":true,"name":null,"roles":["_admin"]}
```

GET /_SESSION

Retorna informações completas sobre o usuário autenticado. Esta informação contém usuário objeto contexto, o método de autenticação e os disponíveis e banco de dados de autenticação.

Parâmetros de Consulta:

básica (boolean) - Aceitar Auth básico, solicitando este recurso. Opcional.

Códigos de status:

200 OK - autenticado com êxito.

401 Unauthorized - Nome de usuário ou senha não foi reconhecido.

Request:

```
GET /_session HTTP/1.1
Host: localhost:5984
Accept: application/json
Cookie: AuthSession=cm9vdDo1MEJDMDQxRDpqb-Ta9QfP9hpdPjHLxNTKg_Hf9w
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: must-revalidate
Content-Length: 175
Content-Type: application/json
Date: Fri, 09 Aug 2013 20:27:45 GMT
Server: CouchDB (Erlang/OTP)
Set-Cookie: AuthSession=cm9vdDo1MjA1NTBDMTQmX2qKt1KDR--GUC80DQ6-Ew_XIW; Version=1; Path=/; HttpOnly

{
  "info": {
    "authenticated": "cookie",
    "authentication_db": "_users",
    "authentication_handlers": [
      "oauth",
      "cookie",
      "default"
    ]
  },
  "ok": true,
  "userCtx": {
    "name": "root",
    "roles": [
      "_admin"
    ]
  }
}
```

APAGAR / session

Fecha a sessão do usuário.

Códigos de status:

200 OK - com sucesso fechar sessão.

401 Unauthorized - O usuário não foi autenticado.

Request:

```
DELETE /_session HTTP/1.1
Accept: application/json
Cookie: AuthSession=cm9vdDo1MjA1NEVGMD01QXNQkqC_0Qmgrk8Fw61_AzDeXw
Host: localhost:5984
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: must-revalidate
Content-Length: 12
Content-Type: application/json
Date: Fri, 09 Aug 2013 20:30:12 GMT
Server: CouchDB (Erlang/OTP)
Set-Cookie: AuthSession=; Version=1; Path=/; HttpOnly

{
  "ok": true
}
```

Autenticação via Proxy

Autenticação via proxy é muito útil no caso de sua aplicação já utiliza algum serviço de autenticação externo e você não deseja duplicar os usuários e seus papéis no CouchDB.

Este método de autenticação permite a criação de um objeto de contexto de usuário para usuário remotamente autenticado. Por padrão, o cliente só precisa passar cabeçalhos específicos para CouchDB com o pedido relacionado:

X-Auth-CouchDB-username: nome de usuário;

X-auth-CouchDB-Roles: lista de funções de usuário separadas por uma vírgula (,);

X-Auth-CouchDB-Símbolo: token de autenticação. Opcional, mas altamente recomendável a força simbólica ser necessária para evitar pedidos de fontes não confiáveis.

Utilize este método de autenticação certificar-se de que a {couch_httpd_auth, proxy_authentication_handler} valor acrescentado à lista dos manipuladores httpd / autenticação ativo:

Request:

```
GET /_session HTTP/1.1
Host: localhost:5984
Accept: application/json
Content-Type: application/json; charset=utf-8
X-Auth-CouchDB-Roles: users,blogger
X-Auth-CouchDB-UserName: foo
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: must-revalidate
Content-Length: 190
Content-Type: application/json
Date: Fri, 14 Jun 2013 10:16:03 GMT
Server: CouchDB (Erlang/OTP)

{
  "info": {
    "authenticated": "proxy",
    "authentication_db": "_users",
    "authentication_handlers": [
      "oauth",
      "cookie",
      "proxy",
      "default"
    ]
  },
  "ok": true,
  "userCtx": {
    "name": "foo",
    "roles": [
      "users",
      "blogger"
    ]
  }
}
```

Note que você não precisa para solicitar sessão a ser autenticada por este método se são fornecidos todos os cabeçalhos HTTP necessárias.

Autenticação OAuth

CouchDB suporta OAuth 1.0 autenticação (RFC 5849). OAuth fornece um método para que os clientes acessem os recursos do servidor sem compartilhar credenciais reais (nome de usuário e senha).

Primeiro, configure o OAuth, através da criação dos consumidores e token com seus segredos e token de ligação para o Real nome de usuário CouchDB.

Provavelmente, não é boa idéia de trabalhar com Curl, vamos usar alguma linguagem de script como Python

```
#!/usr/bin/env python2
from oauth import oauth # pip install oauth
import httplib

URL = 'http://localhost:5984/_session'
CONSUMER_KEY = 'consumer1'
CONSUMER_SECRET = 'sekr1t'
TOKEN = 'token1'
SECRET = 'tokensekr1t'

consumer = oauth.OAuthConsumer(CONSUMER_KEY, CONSUMER_SECRET)
token = oauth.OAuthToken(TOKEN, SECRET)
req = oauth.OAuthRequest.from_consumer_and_token(
    consumer,
    token=token,
    http_method='GET',
    http_url=URL,
    parameters={}
)
req.sign_request(oauth.OAuthSignatureMethod_HMAC_SHA1(), consumer, token)

headers = req.to_header()
headers['Accept'] = 'application/json'

con = httplib.HTTPConnection('localhost', 5984)
con.request('GET', URL, headers=headers)
resp = con.getresponse()
print resp.read()
```

OU Ruby

```
#!/usr/bin/env ruby

require 'oauth' # gem install oauth

URL = 'http://localhost:5984'
CONSUMER_KEY = 'consumer1'
CONSUMER_SECRET = 'sekr1t'
TOKEN = 'token1'
SECRET = 'tokensekr1t'

@consumer = OAuth::Consumer.new CONSUMER_KEY,
                                CONSUMER_SECRET,
                                {:site => URL}

@access_token = OAuth::AccessToken.new(@consumer, TOKEN, SECRET)

puts @access_token.get('/_session').body
```

Ambos os trechos produzem semelhante solicitação e resposta par:

```
GET /_session HTTP/1.1
Host: localhost:5984
Accept: application/json
Authorization: OAuth realm="", oauth_nonce="81430018", oauth_timestamp="1374561749", oauth_consumer_key="cc

HTTP/1.1 200 OK
Cache-Control : must-revalidate
Content-Length : 167
Content-Type : application/json
Date : Tue, 23 Jul 2013 06:51:15 GMT
Server: CouchDB (Erlang/OTP)

{
  "ok": true,
  "info": {
    "authenticated": "oauth",
    "authentication_db": "_users",
    "authentication_handlers": ["oauth", "cookie", "default"]
  },
  "userCtx": {
    "name": "couchdb_username",
    "roles": []
  }
}
```

Solicitamos o recurso `_SESSION` para garantir que a autenticação foi bem sucedida e o nome de usuário CouchDB alvo está correto. Alterar o URL de destino para solicitar recurso necessário.

Alta performance

Maneiras mais rápidas de inserir e dados de consulta com CouchDB. Explicando uma vasta gama de desempenho através de várias técnicas.

Operações em massa resultam em menor sobrecarga, maior rendimento e mais eficiência de espaço. Se você não pode trabalhar com uma grande massa de dados em sua aplicação, nós também vamos descrever outras opções para obter rendimento e benefícios espaciais. Finalmente, descrevemos interface direta com o CouchDB a partir de Erlang, o que pode ser uma técnica útil se você quiser integrar o armazenamento CouchDB com um servidor para não-HTTP protocolos, como SMTP (e-mail) ou XMPP (chat).

Boas Praticas

O resultado: de bons benchmarks exigem carga do mundo real. Simulando a carga é difícil. Erlang tende a ter um melhor desempenho sob carga (especialmente em múltiplos núcleos), por isso temos muitas vezes visto equipamentos de teste que não podem dirigir CouchDB com força suficiente para ver onde ele vai.

Quebrando um único pedido como este e à procura de quanto tempo é gasto em cada componente também é enganoso. Mesmo que apenas uma pequena percentagem do tempo é gasto em seu banco de dados sob carga normal, que não lhe ensinar o que acontecerá durante picos de tráfego. Se todos os pedidos estão batendo na mesma base de dados, então qualquer bloqueio que poderia bloquear muitas solicitações da web. Seu banco de dados pode ter um impacto mínimo sobre o tempo total da consulta, sob carga normal, mas sob carga de pico que pode se transformar em um gargalo, ampliando o efeito do aumento nos servidores de aplicativos. CouchDB pode minimizar este dedicando um processo de Erlang a cada conexão, assegurando que todos os clientes são tratados, mesmo se a latência sobe um pouco.

Performance CouchDB

Foi projetado a partir do zero para atender casos de uso altamente concorrente, que compõem a maioria da carga de aplicativos web. No entanto, às vezes é preciso importar um lote grande de dados em CouchDB ou iniciar transformações através de um banco de dados. Ou talvez nós estamos construindo um aplicativo Erlang personalizado que precisa ligar para CouchDB em um nível inferior do que o HTTP.

Inserções em massa são a melhor maneira de ter gravações seekless. IDs aleatórios forçar procurando depois que o arquivo é maior do que pode ser armazenada em cache. IDs aleatórios também para fazer um arquivo maior, porque em um grande banco de dados que você raramente vai ter vários documentos em uma folha de árvore-B.

Exemplos otimizados: Visualizações e Replicação:

Visualizações carregam um lote de atualizações a partir do disco, passá-los através do mecanismo de exibição em seguida, escrever as linhas vista fora. Cada lote é algumas centenas de documentos, de modo que o escritor pode aproveitar as eficiências em massa que vemos na próxima seção.

Inserções de documentos em massa

O modo mais rápido para importar dados para CouchDB via HTTP é o ponto final `_bulk_docs`. A API documentos em massa aceita uma coleção de documentos em uma única solicitação POST e armazena-los todos para CouchDB em uma única operação de índice.

Massa de docs é a API para usar quando você está importando dados usando uma linguagem de script. Pode ser 10 a 100 vezes mais rápido do que atualizações em massa individuais e é tão fácil de trabalhar com a maioria das línguas.

O principal fator que influencia o desempenho de operações em massa é o tamanho da atualização, tanto em termos de total de dados transferidos, bem como o número de documentos incluídos em uma atualização.

Aqui estão sequenciais inserções de documentos em massa em quatro granularities diferentes, a partir de uma matriz de 100 documentos, através de 1.000, 5.000 e 10.000:

```
bulk_doc_100  
4400 docs  
437.37574552683895 docs/sec
```

```
bulk_doc_1000  
17000 docs  
1635.4016354016355 docs/sec
```

```
bulk_doc_5000  
30000 docs  
2508.1514923501377 docs/sec
```

```
bulk_doc_10000  
30000 docs  
2699.541078016737 docs/sec
```

Você pode ver que lotes maiores produzem melhor desempenho, com um limite superior neste teste de 2.700 documentos / segundo. Com documentos maiores, podemos ver que lotes menores são mais úteis. Para referências, todos os documentos parecido com este: { "foo": "bar"}

Embora 2.700 documentos por segundo é bom, queremos mais poder! Em seguida, vamos explorar a executar documentos em massa em paralelo.

Com um roteiro diferente (usando bash e enrolar com benchbulk.sh no mesmo diretório), estamos inserindo grandes lotes de documentos em paralelo ao CouchDB. Com lotes de 1.000 documentos, 10, em determinado momento, uma média de mais de 10 rodadas, vejo cerca de 3.650 documentos por segundo em um MacBook Pro. Benchbulk também usa IDs seqüenciais.

Vemos que com o uso adequado de documentos a granel e IDs seqüenciais, podemos inserir mais de 3.000 documentos por segundo usando apenas linguagens de script.

Para evitar a sobrecarga de indexação e de disco sincronização associado com documento individual escreve, existe uma opção que permite CouchDB para construir lotes de documentos na memória, rubor-los para o disco quando um determinado limiar foi atingido ou quando acionado pelo usuário. A opção de lote não dá a mesma integridade de dados garante que as atualizações normais fornecem, por isso só deve ser usado quando a perda potencial de atualizações recentes é aceitável.

Como o modo de lote apenas armazena atualizações em memória até que um flush ocorre, as atualizações que estão guardadas no CouchDB diretamente anteriores a um acidente pode ser perdida. Por padrão, o CouchDB libera as atualizações na memória uma vez por segundo, assim, no pior dos casos, a perda de dados ainda é mínimo. Para refletir as garantias de integridade reduzidos quando lote = ok é usado, o código de resposta HTTP é 202 aceitado, em oposição a 201 Criado.

O uso ideal para o modo de lote é para aplicações do tipo de log, onde você tem muitos escritores distribuídos cada armazenamento de eventos discretos para CouchDB. Em um cenário de registro normal, perder algumas atualizações em raras ocasiões, vale a pena o trade-off para o aumento da taxa de transferência de armazenamento.

Há um padrão para armazenamento confiável usando o modo de lote. É o mesmo padrão que é usado quando os dados precisam ser armazenados de forma confiável a vários nós antes de reconhecer o sucesso para o cliente a economizar. Em poucas palavras, o servidor de aplicativos (ou cliente remoto) salva a Couch A utilizando lote = ok, em seguida, relógios

notificações de atualização de Couch B, considerando apenas as economias bem sucedidas quando fluxo de `_changes` do sofá B inclui a atualização relevante.

```
batch_ok_doc_insert
4851 docs
485.00299940011996 docs/sec
```

Esse benchmark JavaScript só recebe cerca de 500 documentos por segundo, seis vezes mais lenta do que a API de documentos em massa. No entanto, tem a vantagem de que os clientes não precisam para construir lotes em massa

Inserções do documento único :

Carga de aplicativo web normal para CouchDB vem na forma de inserções de documentos individuais. Porque cada inserção trata de um cliente diferente, e tem a sobrecarga da totalidade de um pedido HTTP e de resposta, que geralmente tem a mais baixa taxa de transferência para escrita.

Provavelmente o mais lento caso de uso possível para CouchDB é o caso de fazer muitas gravações serializados no banco de dados. Imagine um caso em que cada gravação depende do resultado da escrita anterior de modo que apenas uma escrita pode executar. Isso soa como um mau caso a partir da descrição sozinho. Se você se encontra nesta posição, há provavelmente outros problemas para resolver também.

Olhe para `single_doc_insert` como lento é com full-commit habilitado para quatro ou cinco documentos por segundo! Isso é 100% um resultado do fato de que o Mac OS X tem uma `fsync` real, então ser grato! Não se preocupe; a história completa cometer fica melhor enquanto nos movemos para operações em massa.

Por outro lado, estamos chegando tempos melhores para grandes volumes com atraso cometer off, o que nos permite saber que ajuste para a sua aplicação sempre trará resultados melhores do que seguir um livro de receitas.

Hovercraft

Hovercraft é uma biblioteca para acessar o CouchDB a partir de dentro Erlang. benchmarks hovercraft deve mostrar o desempenho mais rápido possível de subsistemas de disco e de índice do CouchDB, uma vez que evita toda a conexão HTTP e sobrecarga de conversão JSON.

Hovercraft é útil principalmente quando a interface HTTP não permite controle suficiente, ou que seja redundante. Por exemplo, persistindo mensagens instantâneas Jabber para CouchDB pode usar ejabberd e Hovercraft. A maneira mais fácil de criar uma fila de mensagens tolerante a falhas é provavelmente uma combinação de RabbitMQ e Hovercraft.

Hovercraft foi extraído a partir de um projeto de cliente que usou CouchDB para armazenar grandes quantidades de e-mail como anexos de documentos. HTTP não tem um mecanismo fácil para permitir uma combinação de atualizações em massa com anexos binários, por isso usamos Hovercraft para conectar um servidor SMTP Erlang diretamente para CouchDB, para transmitir os anexos diretamente no disco, mantendo a eficiência de atualizações de índice em massa.

Hovercraft inclui uma característica básica benchmarking, e vemos que podemos obter muitos documentos por segundo.

```
> hovercraft:lightning().  
Inserted 100000 docs in 9.37 seconds with batch size of 1000.  
(10672 docs/sec)
```

Trade-Offs

Ferramenta X pode dar-lhe 5 tempos de resposta ms, uma ordem de magnitude mais rápido do que qualquer outra coisa no mercado. A programação é toda sobre os trade-offs, e todo mundo está sujeito às mesmas leis.

Memória

Em vez de fazer cálculos mais e mais, ferramenta X pode ter uma camada de cache bonito que esteja fazendo calculos, armazenando os resultados na memória. Se você está vinculado à CPU, que pode ser bom; se você estiver de memória ligado, talvez não. Um trade-off.

Simultaneidade

As estruturas de dados inteligentes em Ferramenta de X são extremamente rápido quando apenas um pedido de cada vez é processado, e porque é tão rápido na maioria das vezes, ele aparece como se ele iria processar várias solicitações em paralelo. Eventualmente, no entanto, um elevado número de solicitações simultâneas encher a fila de pedidos e tempo de resposta sofre. Uma variação deste é que a Ferramenta de X podem funcionar excepcionalmente bem em uma única CPU ou núcleo, mas não em muitos, deixando seus servidores em marcha lenta.

Confiabilidade

Tornar os dados certa de que está realmente armazenado é uma operação cara. Certificar-se de um banco de dados esteja em um estado consistente e não corrompido é outra. Há duas soluções de compromisso aqui. Em primeiro lugar, buffers armazenar dados na memória antes de cometer-lo no disco para garantir um débito de dados superior. No caso de uma perda de energia ou travamento (de hardware ou software), os dados são ido. Isto pode ou não ser aceitável para a sua aplicação. Em segundo lugar, uma verificação de consistência é necessário para executar após uma falha, e se você tem um monte de dados, isso pode levar dias. Se você pode dar ao luxo de estar off-line, que é OK, mas talvez você não pode permitir isso.

8. Conclusões

Vê-se que o CouchDB é diferente de todos os outros bancos de dados, mas pega emprestado um bom número de paradigmas bem conhecidos e bem entendidos para prover um conjunto de funcionalidades único para seus usuários. O núcleo de armazenamento de dados em JSON em conjunto com o HTTP oferecem um flexível e moderno “data store” com uma engine de consultas bem flexível. Desenhado ao redor do conceito de segurança de dados, CouchDB oferece um armazenamento de dados muito eficiente que é robusto em face de falhas de sistema, falhas de rede, bem como picos repentinos de tráfego. Seu esquema de replicação permite o desenvolvimento descomplicado de aplicações que não ficam 100% do tempo online, sendo inclusive base para outros produtos como o PouchDB. CouchDB foi desenvolvido utilizando Erlang como um projeto Open Source sob uma licença liberal.

9. Referências Bibliográficas

1. Anderson, J. Chris; Slater, Noah; Lehnardt, CouchDB: The Definitive Guide, O'Reilly Media (2010)
2. <http://docs.couchdb.org/en/1.6.1/intro/overview.html?highlight=security>
3. <https://www.infoq.com/articles/apache-couchdb-the-definitive-introduction>
4. <http://guide.couchdb.org/editions/1/en/security.html#validation>
5. <http://docs.couchdb.org/en/1.6.1/api/server/authn.html>