SQL escalável e Lojas NoSQL de dados

Rick Cattell
Cattell.Net Software E-mail:
rick@cattell.net

ABSTRATO

Neste artigo, vamos examinar uma série de SQL e chamado "NoSQL" armazenamentos de dados projetados para dimensionar cargas simples aplicação de estilo OLTP ao longo de muitos servidores. Originalmente motivado por aplicações Web 2.0, esses sistemas são projetados para escalar para milhares ou milhões de usuários fazendo atualizações, bem como lê, em contraste com SGBDs e data warehouses tradicionais. Nós contrastar os novos sistemas em seus mecanismos de modelo de dados, consistência, mecanismos de armazenamento, durabilidade garantias, disponibilidade de apoio de consulta e outras dimensões. Estes sistemas tipicamente sacrificar algumas destas dimensões, por exemplo a consistência de transação de grande base de dados, a fim de alcançar outros, por exemplo, uma maior disponibilidade e escalabilidade.

Nota: As referências bibliográficas para os sistemas não estão listados, mas URLs para mais informações podem ser encontrados na tabela de referências de sistema no final deste artigo. Embargo: As declarações contidas neste documento são baseadas em fontes e documentação que podem não ser confiável, e os sistemas descritos são "alvos móveis", por assim algumas declarações podem estar incorretos. Verificar através de outras fontes antes dependendo informações aqui. No entanto, esperamos que esta pesquisa abrangente é útil! Verifique para futuras correções no cattell.net/datastores web site do autor.

Divulgação: O autor é membro do conselho consultivo técnico da escuna Technologies e tem uma empresa de consultoria aconselhando sobre bases de dados escaláveis.

1. VISÃO GLOBAL

Nos últimos anos uma série de novos sistemas foram projetados para fornecer boa escalabilidade horizontal para operações / banco de dados de gravação de leitura simples, distribuídos por vários servidores. Em contraste, banco de dados tradicional produtos têm relativamente pouca ou nenhuma capacidade de escalar horizontalmente sobre estas aplicações. Este papel analisa e compara os vários novos sistemas. Muitos dos novos sistemas são referidos como "NoSQL" armazenamentos de dados. A definição de NoSQL, que significa "Não só SQL" ou "Não Relacional", não está totalmente acordada. Para os fins deste artigo, sistemas NoSQL geralmente têm seis características principais:

1. a capacidade de escala horizontal "simples operação"rendimento ao longo de muitos servidores,
2. a capacidade de se replicar e para distribuir (partição) dados ao longo de muitos servidores,
3. uma interface de nível de chamada simples ou protocolo (em contraste com uma ligação SQL),
4. um modelo de concorrência mais fraca do que o ACID transações da maioria dos sistemas relacionais (SQL) de banco de dados,
5. utilização eficiente dos índices distribuídos e RAM para de armazenamento de dados, e
6. a capacidade de adicionar dinamicamente novos atributos para registros de dados.

Os sistemas diferem de outras formas, e neste trabalho contrastam essas diferenças. Eles variam em funcionalidade desde o mais simples de hash distribuída, como suportado pelo cache open source memcached popular, para tabelas particionadas altamente escaláveis, como apoiado por BigTable do Google [1].

Na verdade, BigTable, memcached, e Dynamo da Amazon [2] forneceu uma "prova de conceito" que inspirou muitos dos armazenamentos de dados que descrevemos aqui:

- Memcached demonstrado que na memória índices pode ser altamente flexível, distribuir e replicar objectos ao longo de vários nodos.
- Dynamo pioneira a ideia de *eventual consistência* como uma forma de alcançar maior disponibilidade e escalabilidade: os dados coletados não são garantidos para ser up-to-date, mas as atualizações são garantidos para ser propagadas para todos os nós eventualmente.
- BigTable demonstrado que o armazenamento de registro persistente poderia ser escalado para milhares de nós, um feito que a maioria dos outros sistemas de aspirar. Uma característica fundamental dos sistemas NoSQL é escala horizontal "nada compartilhado" - replicação e particionamento de dados ao longo de muitos servidores. Isto lhes permite suportar um grande número de operações simples de leitura / gravação por segundo. Esta carga operação simples é tradicionalmente chamado OLTP (processamento de transações online), mas também é comum em aplicações web modernas

Os sistemas NoSQL descrito aqui geralmente não fornecem ACID propriedades transacionais: atualizações, eventualmente, são propagadas, mas existem garantias limitados sobre a consistência das leituras. Alguns autores sugerem uma sigla "BASE" em contraste com a sigla "ACID":

- BASE = Basicamente Disponível, Soft Estado, Eventualmente consistente
- ÁCIDO = atomicidade, consistência, isolamento, e Durabilidade

A ideia é que, dando-se restrições ACID, pode-se conseguir muito mais elevado desempenho e escalabilidade.

No entanto, os sistemas diferem em quanto eles desistem. Por exemplo, a maioria dos sistemas chamam a si mesmos “eventualmente consistente”, o que significa que as atualizações são eventualmente propagadas para todos os nós, mas muitos deles fornecer mecanismos para algum grau de consistência, tais como controle de concorrência multi-versão (MVCC). Os defensores da NoSQL costumam citar CAP teorema de Eric Brewer [4], que afirma que um sistema pode ter apenas dois dos

três de as seguintes propriedades:
consistência, disponibilidade, e partição-tolerância. Os sistemas NoSQL geralmente desistir consistência. No entanto, os trade-offs são complexos, como veremos. Novas SGBDs relacionais também foram introduzidas para proporcionar um melhor escalonamento horizontal para OLTP, quando comparado com RDBMSs tradicional.

depois de examinar
os sistemas NoSQL, vamos olhar para estes sistemas SQL e comparar os pontos fortes das abordagens. Os sistemas SQL nos esforçamos para oferecer escalabilidade horizontal sem abandonar as transações SQL e ácido. Vamos discutir os trade-offs aqui. Neste artigo, vamos nos referir a ambos os sistemas NoSQL como novo SQL e *armazenamentos de dados,* uma vez que o termo “sistema de banco de dados” é amplamente utilizado para se referir a SGBDs tradicional.

No entanto, ainda vamos usar o termo
“Banco de dados” para se referir aos dados armazenados nestes sistemas. Todas as lojas de dados têm alguma unidade administrativa que você chamaria de um banco de dados: dados podem ser armazenados em um arquivo, ou em um diretório, ou através de algum outro mecanismo que define o âmbito dos dados utilizados por um grupo de aplicações. Cada banco de dados é uma ilha em si mesmo, mesmo se o banco de dados é particionado e distribuído por várias máquinas:

não há “federado
conceito de banco de dados”nestes sistemas (como acontece com alguns bancos de dados relacionais e orientada para o objecto), permitindo que várias bases de dados separadamente administrados a aparecer como um. A maioria dos sistemas permitem particionamento horizontal de dados, armazenando registros em servidores diferentes, de acordo com alguma chave; isso é chamado de “fragmentação”. Alguns dos sistemas também permitem o particionamento vertical, onde as partes de um único registro são armazenados em servidores diferentes.

1.1 Âmbito do presente Livro

Antes de prosseguir, alguns esclarecimentos são necessários para definir “horizontal escalabilidade”e“simples
operações". Estes definem o foco deste artigo. Por “operações simples”, nos referimos a pesquisas de chave, lê e escreve de um registro ou um pequeno número de registros. Isso está em contraste com consultas complexas ou junta, acesso à leitura em sua maioria, ou outras cargas de aplicativos. Com o advento da web, especialmente Web 2.0 locais onde milhões de usuários podem ler e escrever dados, escalabilidade para operações de banco de dados simples tornou-se mais importante. Por exemplo, os pedidos podem pesquisar e atualizar bancos de dados de vários servidores de correio eletrônico, perfis pessoais, publicações web, wikis, cliente
registros, registros de namoro online, anúncios classificados, e muitos outros tipos de dados. Todos estes geralmente se ajustar ao definição de “operação simples” aplicações: ler ou escrever um pequeno número de registros relacionados em cada operação.

O termo “escalabilidade horizontal” significa a capacidade de distribuir os dados e a carga destas operações simples ao longo de muitos servidores, sem RAM ou disco compartilhado entre os servidores. escala horizontal difere da escala “vertical”, onde um sistema de banco de dados utiliza muitos núcleos e / ou CPUs que compartilham memória RAM e discos. Alguns dos sistemas que descrevemos fornecer escalabilidade vertical e horizontal, e o uso eficaz de múltiplos núcleos é importante, mas nosso foco principal é sobre escalabilidade horizontal, porque o número de núcleos que podem compartilhar memória é limitado e escala horizontal geralmente revelar menos caros, usando servidores de commodities. Note-se que horizontal e vertical *particionamento*

não estão relacionados com a horizontal e vertical *dimensionamento,*
exceto que eles são úteis para o dimensionamento horizontal.

1,2 Sistemas além do nosso escopo

Alguns autores têm utilizado uma definição ampla de NoSQL, incluindo qualquer sistema de banco de dados que não é relacional. Especificamente, eles incluem:

- *sistemas de banco de dados Graph:* Neo4j e OrientDB fornecer armazenamento distribuído eficiente e consultas de um gráfico de nodos com referências entre eles.
- *sistemas de banco de dados orientado a objetos:* SGBD orientada para objectos (por exemplo, Versant) também proporcionam armazenamento distribuído eficiente de um gráfico de objectos, e materializam
  esses objetos como programação
  objetos de linguagem.
- *Distribuído lojas orientadas a objeto:* Muito semelhante SGBDs orientada a objeto para, sistemas como GemFire distribuir gráficos de objetos na memória em vários servidores.

Estes sistemas são uma boa escolha para aplicações que deve fazer rápida e extensa referência de acompanhamento, especialmente onde os dados se encaixa na memória. integração de linguagem de programação também é valioso. Ao contrário dos sistemas NoSQL, estes sistemas geralmente fornecem transações ACID. Muitos deles oferecem escalonamento horizontal para referência de acompanhamento e decomposição consulta distribuída, bem. Devido a limitações de espaço, no entanto, omitimos estes sistemas a partir de nossas comparações.

As aplicações e os necessários
otimizações para escalar para esses sistemas diferem dos sistemas que cobrimos aqui, onde chave pesquisas e operações simples predominam sobre referência-
seguinte e comportamento dos objectos complexo. É possível estes sistemas pode escalar em operações simples assim, mas isso é um assunto para um artigo futuro, e à prova através de benchmarks.

sistemas de banco de dados de armazenamento de dados fornecem escalabilidade horizontal, mas também estão fora do escopo deste artigo. aplicações de armazenamento de dados são diferentes em maneiras importantes:

- Eles executam consultas complexas que coletam e juntar informações de muitas tabelas diferentes.
- A proporção de leituras para gravações é elevado, ou seja, o banco de dados é somente leitura ou leitura principalmente. Existem sistemas existentes de armazenamento de dados que escalam bem horizontalmente.

Uma vez que os dados são atualizados frequentemente, é possível organizar ou replicar o banco de dados de maneiras que fazem escala possível.

1.3 Modelo de Dados Terminologia

Ao contrário relacional (SQL) SGBDs, a terminologia utilizada pelos armazenamentos de dados NoSQL é muitas vezes inconsistente. Para os fins deste artigo, precisamos de uma maneira consistente para comparar os modelos de dados e funcionalidade. Todos os sistemas descritos aqui fornecem uma maneira de armazenar valores escalares, como números e strings, assim como BLOBs. Alguns deles também fornecem uma maneira de armazenar mais complexas valores aninhados ou de referência. Os sistemas de todos os conjuntos de lojas de pares atributo-valor, mas usar diferentes estruturas de dados, especificamente:

- A "tupla" é uma linha em uma tabela relacional, onde nomes de atributos são pré-definidos em um esquema e os valores devem ser escalar. Os valores são referenciado pelo nome de atributo, ao contrário de uma matriz ou lista, onde eles são referenciados por posição ordinal.
- A "documento" permite que os valores sejam aninhados documentos ou listas, bem como valores escalares, e os nomes de atributo são definidos de forma dinâmica para cada documento em tempo de execução. Um documento difere de uma tupla em que os atributos não são definidos em um esquema global, e são permitidas esta ampla gama de valores.
- Um "registro extensível" é um híbrido entre uma tupla e um documento, onde as famílias de atributos são definidos em um esquema, mas novos atributos podem ser adicionados (dentro de uma família atributo) em uma base per-registro. Os atributos podem ser valorizados-list.
- Um "objecto" é análogo a um objecto em programação línguas, mas sem a métodos processuais. Os valores podem ser referências ou objetos aninhados.

1,4 Categorias Data Store

Neste trabalho, os armazenamentos de dados são agrupados de acordo com seu modelo de dados:

- *Lojas de valor-chave:* Estes valores armazenar sistemas e um índice de encontrá-los, com base em uma chave programmer- definido.
- *Lojas de documentos:* Estes documentos armazenar sistemas, como apenas definido. Os documentos são indexados e um mecanismo de consulta simples é fornecido.
- *Extensíveis lojas de discos:* Esses registros extensíveis armazenar sistemas que pode ser particionado verticalmente e horizontalmente em todos os nós. Alguns papéis chamar estas "lojas de coluna de largura".
- *Bancos de dados relacionais:* Estes sistemas armazenam (e indexação e de consulta) tuplas. O novo RDBMSs que fornecem escalabilidade horizontal são abordados neste papel.

armazenamentos de dados nestas quatro categorias são cobertos na próxima quatro seções, respectivamente. Nós vamos então resumir e comparar os sistemas.

2. STORES de valores-chave

Os armazenamentos de dados mais simples utilizar um modelo de dados semelhante ao memcached populares distribuídos cache na memória, com um único índice de chave-valor para todos os dados. Vamos chamar esses sistemas *lojas de chave-valor*. Ao contrário memcached, estes sistemas geralmente proporcionar um mecanismo de persistência e funcionalidade adicional, bem como: replicação, de versão, de bloqueio, as transacções, triagem, e / ou outras características. A interface do cliente fornece inserções, exclusões e pesquisas de índice. Como memcached, nenhum destes sistemas oferecem índices secundários ou chaves.

2.1 projeto Voldemort

Projeto Voldemort é uma loja de valor-chave avançado, escrito em Java. É open source, com substancial contribuições do LinkedIn. Voldemort fornece controle de concorrência multi-versão (MVCC) para atualizações. Ele atualiza as réplicas de forma assíncrona, por isso não garante dados consistentes. No entanto, ele pode garantir uma visão up-to-date se você ler uma maioria de réplicas.

Voldemort suporta bloqueio otimista para atualizações multi-recordes consistentes: se as atualizações conflito com qualquer outro processo, eles podem ser desfeitas. relógios vetoriais, como usado em Dynamo [3], fornecer uma ordenação em versões. Você também pode especificar qual versão você deseja atualizar para a venda e operações de exclusão. Voldemort suporta fragmentação automática de dados.

hashing consistente é usado para distribuir dados em torno de um anel de nodos: dados hash para o nó K é replicado no nó K + 1 ... K + n, onde n é o número desejado de cópias extra (muitas vezes n = 1).

Usando boa sharding técnica, deve haver muitos mais nós "virtuais" do que nós físicos (servidores). Uma vez particionamento de dados é criado, o seu funcionamento é transparente. Os nós podem ser adicionados ou removidos a partir de um agrupamento do banco de dados, e o sistema se adapta automaticamente. Voldemort detecta automaticamente e recupera nós falhos.

Voldemort pode armazenar dados na RAM, mas também permite ligar um motor de armazenamento. Em particular, ele suporta um Berkeley DB e Acesso Aleatório mecanismo de armazenamento de arquivos. Voldemort suporta listas e registros, além de valores escalares simples.

2.2 Riak

Riak é escrito em Erlang. Foi open-source por Basho em meados de 2009. Basho descreve alternadamente Riak como uma "loja do valor-chave" e "armazenamento de documentos". Vamos classificá-lo como um armazenamento de chaves-valor avançado aqui, porque não tem características importantes de armazenamentos de documentos, mas (e Voldemort) tem mais funcionalidades do que as outras lojas de valor-chave:

- objetos Riak pode ser obtido e armazenado em formato JSON e, portanto, pode ter vários campos (como documentos), e objetos podem ser agrupados em baldes, como os conjuntos suportados pelas lojas de documentos, com permitidos campos / necessários definidos em uma base por balde.
- O Riak não suporta índices em todos os campos, exceto a chave primária. A única coisa que você pode fazer com os campos não-primária é buscar e armazená-los como parte de um objeto JSON. Riak carece dos mecanismos de consulta das lojas de documentos; a única pesquisa que você pode fazer é na chave primária. Riak suporta replicação de objetos e sharding por hash na chave primária. Ele permite que os valores de réplica a ser temporariamente inconsistente. Consistência é ajustável, especificando quantas réplicas (em nós diferentes) deve responder para uma leitura bem sucedida e quantos devem responder para uma gravação bem-sucedida. Isto é por ler e per-write, de modo diferentes partes de um aplicativo pode escolher diferentes trade-offs.

Como Voldemort, Riak utiliza um derivado do MVCC onde os relógios vetoriais são atribuídos quando os valores são atualizados. relógios vetor pode ser usado para determinar quando os objetos são descendentes diretos do outro ou um pai comum, por isso Riak pode muitas vezes dados de auto-reparo que descobre estar fora de sincronia.

A arquitetura Riak é simétrico e simples. Como Voldemort, ele usa hashing consistente. Não há nó distinto para controlar o status do sistema: os nós de utilizar um protocolo de fofocas de rastrear quem está vivo e quem tem que dados e em qualquer nó pode atender uma solicitação do cliente. Riak também inclui um mapa / reduzir mecanismo para dividir trabalho ao longo de todos os nós de um cluster. A interface do cliente para Riak é baseada em solicitações HTTP RESTful. RESTO (REpresentational State Transfer) utiliza uniformes, chamadas apátrida, armazenáveis, cliente-servidor. Há também uma interface de programação para Erlang, Java e outras linguagens.

A parte de armazenamento de Riak é "pluggable": os pares chave-valor pode ser na memória, em tabelas ETS, na DETS tabelas, ou em tabelas Osmos. ETS, DETs e Osmos mesas estão todas implementadas no Erlang, com diferentes desempenho e propriedades.

Uma característica única de Riak é que ele pode armazenar "ligações" entre objetos (documentos), por exemplo, para vincular objetos para que os autores os objetos para os livros que escreveu. Ligações reduzir a necessidade de índices secundários, mas ainda não há uma maneira de fazer consultas alcance. Aqui está um exemplo de um objeto de Riak descrito em JSON:

```
{  "Balde": "clientes", "chave":
 "12345", "objeto": {

        "Nome": "Sr. Smith.",
    "Telefone":"415-555-6524" } "links": [

      [ "Vendas", "Mr salesguy.", "SalesRep"], [
  "cust-encomendas", "12345", "ordens"]] "vclock":
  "opaco-riak-vclock", "lastmod": "Mon , 3 de agosto de 2009
  18:49:42 GMT"}
```

Note-se que a chave primária é distinto, enquanto os outros campos são parte de uma porção "objecto". Além disso, note que o balde, relógio do vetor, e data de modificação é especificado como parte do objecto, e as ligações a outros objectos são suportados.

2.3 Redis

O armazenamento de dados de chave-valor Redis começou como um projeto de uma pessoa, mas agora tem vários colaboradores como BSD licenciados open source. Ele é escrito em servidor C. A Redis é acessado por um protocolo fio implementado em várias bibliotecas do cliente (que devem ser atualizados quando as mudanças de protocolo). O lado do cliente faz o hashing distribuiu mais servidores. Os dados servidores armazenar na memória RAM, mas os dados podem ser copiados para o disco para backup ou sistema de desligamento. desligamento do sistema podem ser necessários para adicionar mais nós.

Como as outras lojas de chave-valor, Redis implementa inserir, excluir e operações de pesquisa. Como Voldemort, ele permite que listas e conjuntos para ser associado com uma chave, não apenas uma bolha ou string. Ele também inclui lista e conjunto operações.

Redis faz atualizações atômicas, bloqueando, e faz replicação assíncrona. É relatado para apoiar cerca de 100K fica / sets por segundo em um servidor 8-core.

2.4 scalaris

Scalaris é funcionalmente semelhante ao Redis. Ele foi escrito em Erlang no Instituto Zuse em Berlim, e é open source. Na distribuição de dados sobre nós, permite intervalos de chaves a ser atribuído a nós, ao invés de simplesmente hash para nós. Isto significa que uma consulta em um intervalo de valores não precisa ir a cada nó, e também pode permitir um melhor balanceamento de carga, dependendo de distribuição de chaves.

Como as outras lojas de valor-chave, ele suporta inserir, excluir e pesquisa. Ele faz a replicação síncrona (cópias devem ser atualizadas antes que a operação está completo) para que os dados é garantida para ser consistente. Scalaris também suporta transações com ácido Propriedades em vários objectos. Os dados são armazenados na memória, mas a replicação e recuperação de falhas de nó oferece durabilidade de as atualizações. No entanto, uma falha de energia multi-nó causaria perda desastrosa de dados, eo limite de memória virtual define um tamanho máximo de banco de dados.

Scalaris lê e escreve deve ir para a maioria das réplicas antes de uma operação concluída. Scalaris utiliza um anel de nós, uma estratégia incomum distribuição e de replicação que requer log (N) lúpulo para ler / escrever um par de valores de chave-.

2,5 Gabinete de Tóquio

Tokyo Cabinet / Tokyo Tyrant foi um projeto sourcefourge.net, mas agora é licenciado e mantido pelo FAL Labs. Tokyo Cabinet é o servidor back-end, Tokyo Tyrant é uma biblioteca cliente para acesso remoto. Ambos são escritos em C.

Existem seis variações diferentes para o servidor Tokyo Cabinet: índices hash na memória ou no disco, B- árvores na memória ou no disco, mesas de registro de tamanho fixo e tabelas de registro de comprimento variável. os motores obviamente, diferem em suas características de desempenho, por exemplo, os registos de comprimento fixo permitir pesquisas rápidas. Há pequenas variações na API suportada por estes motores, mas todos eles apoiar comum obter operações / set / atualizar. A documentação é um pouco incerto, mas eles dizem apoiar bloqueio, transações ACID, um tipo de dados matriz binária, e operações de atualização mais complexas para atualizar atomicamente um número ou concatenar a uma corda. Eles apoiam assíncrona replicação com dupla ou mestre senhor de escravos. Recuperação de um nó não é manual, e não há nenhuma fragmentação automática.

2.6 Memcached, MemBrain e Membase

O open-source memcached distribuídos em memória sistema de indexação tenha sido reforçada por Schooner Tehnologies e Membase, para incluir recursos análogas às outras lojas de valor-chave: persistência, de replicação, alta disponibilidade, de crescimento dinâmico, backup e assim por diante. Sem persistência ou replicação, memcached realmente não qualificar como um "armazenamento de dados". No entanto, MemBrain e Membase certamente fazer, e esses sistemas também são compatíveis com aplicações memcached existentes. Esta compatibilidade é uma característica atraente, dado que memcached é amplamente utilizado; usuários memcached que exigem recursos mais avançados podem facilmente atualizar para Membase e MemBrain.

O sistema Membase é open source, e é apoiado pela empresa Membase. Sua característica mais atraente é, provavelmente, a sua capacidade de adicionar elasticamente ou remover servidores em um sistema em execução, a movimentação de dados e dinamicamente redirecionando solicitações no mesmo período. A elasticidade na maioria dos outros sistemas não é tão conveniente.

MemBrain é licenciado por servidor, e é apoiado por Schooner Technologies. Sua característica mais atraente é, provavelmente, a sua excelente ajuste para memória flash. Os ganhos de memória flash de desempenho não será adquirida em outros sistemas de tratamento de flash como um disco rígido mais rápido; é importante que o flash deleite sistema como um verdadeiro "terceiro nível", diferente de memória RAM e disco.

Para exemplo, muitos sistemas têm sobrecarga substancial no buffer e cache páginas de disco rígido; isto é sobrecarga desnecessária com flash. Os resultados do benchmark no site show de Schooner muitos *vezes* melhor desempenho do que um número de concorrentes, especialmente quando os dados transborda RAM.

2.7 Resumo

Todas as lojas de valor-chave apoiar inserir, excluir e operações de pesquisa. Todos estes sistemas fornecem escalabilidade através da distribuição chave sobre nós. Voldemort, Riak, Tokyo Cabinet, e sistemas memcached aprimorados pode armazenar dados na RAM ou no disco, com armazenamento de add-ons. Os outros armazenar dados na RAM, e fornecer disco como backup, ou depender de replicação e recuperação para que um backup não é necessário. Scalaris e sistemas memcached aprimorados usar a replicação síncrona, o uso restante assíncrona. Scalaris e Tóquio Gabinete implementar transações, enquanto os outros não.

Voldemort e Riak usar controlo concorrência de múltiplas versões (MVCC), os outros utilizam fechaduras. MemBrain e Membase são construídos no sistema memcached popular, adicionando persistência, replicação e outros recursos. compatibilidade com memcached dar a estes produtos uma vantagem.

3. LOJAS DE DOCUMENTOS

Como discutido na primeira seção, lojas documento de suporte de dados mais complexas do que as lojas de chave-valor. O termo "armazenamento de documentos" pode ser confuso: enquanto estes sistemas podem armazenar "documentos" no sentido tradicional (artigos, arquivos do Microsoft Word, etc.), um documento nesses sistemas podem ser qualquer tipo de "objeto pointerless", de acordo com nossa definição na Seção 1. Ao contrário das lojas de valor-chave, estes sistemas geralmente suportam índices secundários e vários tipos de documentos (objetos) por banco de dados e documentos aninhados ou listas. Como outros sistemas NoSQL, o

lojas de documentos não fornecem ACID propriedades transacionais.

3.1 SimpleDB

SimpleDB é parte da oferta de computação em nuvem de propriedade da Amazon, juntamente com a sua Elastic Compute Cloud (EC2) e sua Simple Storage Service (S3) em que SimpleDB é baseado. SimpleDB tem sido em torno desde 2007. Como o nome sugere, o seu modelo é simples: SimpleDB tem Select, Delete, GetAttributes e operações PutAttributes em documentos. SimpleDB é mais simples do que outras lojas de documentos, uma vez que não permite que documentos aninhados.

Como a maioria dos sistemas que discutimos, SimpleDB suporta consistência eventual, não transacional consistência. Como a maioria dos outros sistemas, ele faz a replicação assíncrona.

Ao contrário de armazenamentos de dados de valor-chave e, como as outras lojas de documentos, SimpleDB suporta mais de um agrupamento em um banco de dados: os documentos são colocados em domínios, que suportam vários índices. Você pode enumerar domínios e seus metadados. Seleccione operações estão em um domínio, e especificar um conjunto de restrições sobre atributos, basicamente na forma:

Seleccione <atributos> de <domain>, onde
<Lista de restrições de valor atributo>

Diferentes domínios podem ser armazenados em diferentes nós da Amazônia.

índices de domínio são atualizados automaticamente quando os atributos de qualquer documento são modificados. Não está claro a partir da documentação se SimpleDB seleciona automaticamente qual atribui ao índice, ou se ele indexa tudo. Em ambos os casos, o usuário não tem escolha, e o uso dos índices é automática em processamento de consulta SimpleDB.

O SimpleDB não particionar automaticamente dados sobre os servidores. Alguns escala horizontal pode ser alcançar através da leitura de qualquer das réplicas, se você não se importa em ter a versão mais recente. Gravações não escala, no entanto, porque eles devem ir de forma assíncrona para todas as cópias de um domínio. Se os clientes querem melhor dimensionamento, devem fazê-lo manualmente por sharding si. SimpleDB é um "pay as you go" solução proprietária da Amazônia. Atualmente restrições integradas, alguns dos quais são bastante limitantes: um tamanho máximo de domínio de 10 GB, um limite de 100 domínios activos, um segundo limite de 5 em consultas, e assim por diante. Amazônia não licenciar fonte SimpleDB ou código binário para rodar em seus próprios servidores.

SimpleDB tem a vantagem de apoio Amazon e documentação.

3.2 CouchDB

CouchDB foi um projeto Apache desde o início 2008. Ele é escrito em Erlang.

Uma "coleção" CouchDB de documentos é semelhante a um domínio SimpleDB, mas o modelo de dados CouchDB é mais rico. Colecções incluem o único esquema no CouchDB, e índices secundários devem ser explicitamente criado em campos em coleções. Um documento tem valores de campo que podem ser escalar (texto, numérico, ou boolean) ou composto (um documento ou lista). As consultas são feitas com o CouchDB chama de "pontos de vista", que são definidos com Javascript para especificar restrições de campo. Os índices são árvores-B, de modo que os resultados das pesquisas podem ser encomendados ou intervalos de valores. Consultas pode ser distribuído em paralelo ao longo de vários nós utilizando um MAP- reduzir mecanismo.

No entanto, a visão de CouchDB mecanismo coloca mais carga sobre os programadores do que uma linguagem de consulta declarativa.

Como SimpleDB, CouchDB alcança escalabilidade através assíncrona replicação, não através sharding. Lê pode ir a qualquer servidor, se você não se importa em ter os valores mais recentes e atualizações devem ser propagadas para todos os servidores. No entanto, um novo projeto chamado CouchDB salão foi construído para fornecer sharding em cima do CouchDB, veja:

http://code.google.com/p/couchdb-lounge/ Como SimpleDB, CouchDB não garante consistência. Ao contrário de SimpleDB, cada cliente faz ver uma vista auto-consistente do banco de dados, com lê repetível:

CouchDB implementos multi-versão controle de concorrência em documentos individuais, com um ID de seqüência que é criada automaticamente para cada versão de um documento. CouchDB irá notificar um aplicativo se alguém actualiza o documento, uma vez que foi buscado. O aplicativo pode então tentar combinar as atualizações, ou pode simplesmente repetir a sua actualização e substituir.

CouchDB também oferece durabilidade em falha do sistema. Todas as atualizações (documentos e índices) são liberadas para disco em comprometer, por escrito, ao final de um arquivo. (Isto significa que a compactação periódica é necessária.) Por padrão, ele libera para o disco depois de cada atualização do documento. Juntamente com o mecanismo MVCC, durabilidade do CouchDB proporciona assim semântica ACID no nível do documento.

Clientes chamar CouchDB através de uma interface RESTful. Existem bibliotecas para várias linguagens (Java, C, PHP, Python, LISP, etc) que convertem a API nativa chama para as chamadas de descanso para você. CouchDB tem algumas funcionalidades básicas administração de banco de dados também.

3,3 MongoDB

MongoDB é um repositório de documentos open source GPL escrito em C ++ e apoiado por 10gen. Ele tem algumas semelhanças com o CouchDB: ele fornece índices em coleções, é lockless, e fornece um mecanismo de consulta de documentos. No entanto, existem diferenças importantes:

- MongoDB suportes automático sharding, distribuição de documentos mais servidores.
- Replicação em MongoDB é usado principalmente para failover, não para (leitura suja) escalabilidade como no CouchDB. O MongoDB não fornece a consistência global de uma DBMS tradicional, mas você pode obter a consistência local na cópia primária up-to-date de um documento.

- MongoDB oferece suporte a consultas dinâmicas com o uso automático de índices, como RDBMSs. Em CouchDB, os dados são indexados e pesquisados por escrito Map-Reduce vistas.
- CouchDB oferece MVCC em documentos, enquanto MongoDB fornece operações atômicas sobre campos. operações atômicas sobre campos são fornecidos como segue:
- O comando actualização suporta “modificadores” que facilitam mudanças atômicas para valores individuais: $ set define um valor, $ inc incrementa um valor, $ impulso acrescenta um valor para uma matriz, $ pushAll anexa vários valores para uma matriz, $ puxar remove um valor a partir de uma matriz, e $ pullAll remove vários valores a partir de uma matriz.

  Uma vez que estas alterações normalmente ocorrem “no lugar”, eles evitam o sobrecarga de uma viagem de volta para o servidor.
- Há um “update se a corrente” convenção para mudar um documento somente se valores de campo corresponder a um dado valor anterior.

- MongoDB suporta um comando findAndModify para executar uma atualização atômica e retornar imediatamente o documento atualizado. Isso é útil para a implementação de filas e outras estruturas de dados que necessitam de atomicidade.

índices MongoDB são explicitamente definidas usando uma chamada ensureIndex, e quaisquer índices existentes são automaticamente usado para o processamento de consultas. Para encontrar todos os produtos lançados no ano passado custando menos de US $ 100 você poderia escrever:

```
db.products.find (
  {Lançado: {$ gte: new Date (2009, 1, 1,)}, preço { '$
  lte': 100},})
```

Se os índices são definidos nos campos consultados, MongoDB irá utilizá-los automaticamente. MongoDB também suporta Map-Reduce, que permite agregações complexas através de documentos.

MongoDB armazena dados em um formato JSON-like binário chamado BSON. BSON suporta boolean, integer, float, data, corda e tipos binários. drivers de cliente codificar estrutura de dados de documentos do idioma local (geralmente um dicionário ou matriz associativa) em BSON e enviá-lo através de uma conexão socket para o servidor MongoDB (em contraste com CouchDB, que envia JSON como texto sobre um interface REST HTTP). MongoDB também suporta uma especificação GridFS para grandes objetos binários, por exemplo.

imagens e vídeos. Estes são armazenados em blocos que podem ser transmitidos de volta ao cliente para a entrega eficiente. MongoDB suporta replicação mestre-escravo com failover e recuperação automática. Replication (e recuperação) é feito ao nível dos cacos. As coleções são automaticamente sharded através de uma chave fragmento definido pelo utilizador. Replicação é assíncrona para maior desempenho, portanto, algumas atualizações podem ser perdidos em um acidente.

3,4 Terrastore

Outra loja recente documento é Terrastore, que é construído sobre o produto agrupamento Java VM Terracotta distribuído. Como muitos dos outros sistemas NoSQL, acesso do cliente ao Terrastore é construído sobre operações HTTP para buscar e armazenar dados. APIs cliente Java e Python também foram implementadas. Terrastore partições automaticamente os dados sobre nós do servidor, e pode redistribuir automaticamente os dados quando os servidores são adicionados ou removidos. Como MongoDB, ele pode executar consultas com base em um predicado, incluindo consultas de intervalo e, como CouchDB, que inclui um mapa / reduzir mecanismo para seleção mais avançada e agregação de dados.

Como os outros bancos de dados de documentos, Terrastore é sem esquema, e não fornece transações ACID. Como MongoDB, fornece consistência em uma base documento per-: uma leitura será sempre buscar a versão mais recente de um documento.

Terrastore suporta replicação e failover a um hot standby.

3.5 Resumo

As lojas de documentos estão sem esquema, com exceção de atributos (que são simplesmente um nome, e não são pré-especificada), colecções (que são simplesmente um agrupamento de documentos), e os índices definidos em colecções (definidas explicitamente, exceto com SimpleDB). Existem algumas diferenças em seus modelos de dados, por exemplo SimpleDB não permite que documentos aninhados. As lojas de documentos são muito semelhantes, mas usar uma terminologia diferente. Por exemplo, um banco de dados SimpleDB Domínio = CouchDB = MongoDB Coleção = Terrastore Bucket.

SimpleDB chama documentos “itens”, e um atributo é um campo em CouchDB, ou uma chave no MongoDB ou Terrastore. Ao contrário das lojas de valor-chave, as lojas de documentos fornecem um mecanismo para consultar coleções com base em múltiplos constrangimentos valor do atributo.

Contudo, O CouchDB não suporta uma linguagem de consulta não-processual: ele coloca mais trabalho do programador e requer a utilização explícita de índices. As lojas de documentos em geral não fornecem bloqueios explícitos, e têm mais fracas propriedades de simultaneidade e atomicidade do que bancos de dados tradicionais ACID-compliant.

Eles diferem em quanto o controle de concorrência que eles fornecem.

Os documentos podem ser distribuídos mais de nós em todos os sistemas, mas a escalabilidade é diferente. Todos os sistemas podem alcançar escalabilidade através da leitura (potencialmente) réplicas out-of-date.

MongoDB e Terrastore pode obter escalabilidade sem que o compromisso, e pode escalar escritas, bem como, através de fragmentação automática e operações atômicas sobre documentos. CouchDB pode ser capaz de alcançar este write-escalabilidade com a ajuda do novo código CouchDB Sala de estar.

Um adendo de última hora como este trabalho vai para a imprensa: o CouchDB e e empresas Membase têm agora fundido, para formar Couchbase. Eles pretendem fornecer um "melhor de ambos" merge de seus produtos, por exemplo, com modelo de dados mais rica do CouchDB, bem como a velocidade e elástica escalabilidade de Membase. Veja Couchbase.com para mais informações.

4. Lojas de discos EXTENSÍVEIS

As lojas de discos extensíveis parecem ter sido motivados pelo sucesso do Google com BigTable. Seu modelo de dados básica é linhas e colunas, e seu modelo básico escalabilidade é dividir linhas e colunas sobre vários nós:

- As linhas são divididas entre nós através de sharding na chave primária. Eles normalmente dividido pelo intervalo em vez de uma função hash. Isto significa que as consultas sobre faixas de valores não tem que ir para cada nó.

- Colunas de uma tabela são distribuídos ao longo de vários nós utilizando "grupos de colunas". Estes podem parecer como uma nova complexidade, mas grupos de colunas são simplesmente uma maneira para o cliente para indicar quais colunas são melhor armazenados juntos.

Como observado anteriormente, estes dois tabiques (horizontal e vertical) pode ser utilizado simultaneamente na mesma tabela. Por exemplo, se uma tabela de clientes é dividido em três grupos de colunas (por exemplo, separando o nome do cliente / endereço de informação financeira e login), em seguida, cada um dos três grupos de colunas é tratada como uma tabela separada para os fins de sharding as linhas por ID do cliente: os grupos de colunas para um cliente pode ou não estar no mesmo servidor.

Os grupos de colunas deve ser pré-definido com as lojas de discos extensíveis. No entanto, isso não é um grande constrangimento, como novos atributos podem ser definidos em qualquer momento. Linhas são análogos aos documentos: eles podem ter um número variável de atributos (campos), os nomes de atributos devem ser únicos, as linhas são agrupadas em coleções (tabelas), e os atributos de uma linha individual pode ser de qualquer tipo. (No entanto, note que CouchDB e MongoDB apoiar objetos aninhados, enquanto as lojas de discos extensíveis geralmente suportam apenas tipos escalares.)

Embora a maioria das lojas de discos extensíveis foram modeladas após BigTable, parece que nenhuma das lojas dos registros extensíveis chegar perto a escalabilidade do BigTable no presente. BigTable é usado para muitas finalidades (pense dos muitos serviços do Google oferece, não apenas busca na web).

Vale a pena ler o papel BigTable [1] para o fundo sobre os desafios com escala.

4.1 HBase

HBase é um projeto Apache escrito em Java. Isto é modelada diretamente após BigTable:

- HBase usa o sistema de arquivos distribuído Hadoop no lugar do sistema de arquivos do Google. Ela coloca atualizações na memória e periodicamente escreve-los para arquivos no disco.

- As atualizações de ir para o final de um arquivo de dados, para evitar a procura. Os arquivos são periodicamente compactado. As atualizações também ir para o fim de uma gravação de log frente, para executar a recuperação Se um servidor falhas.

- operações de linha são atômicas, com bloqueio de nível de linha e transações. Há suporte opcional para transações com escopo mais amplo. estes uso controle de simultaneidade otimista, abortando se houver um conflito com outras atualizações.

- Separação e distribuição são transparentes; não há hashing do lado do cliente ou keyspace fixa como em alguns sistemas NoSQL. Há suporte mestre múltipla, para evitar um ponto único de falha. apoio MapReduce permite que as operações sejam distribuídos de forma eficiente.

- B-trees de HBase permitir consultas alcance rápido e classificação.

- Há uma API Java, a API Thrift, e API REST. suporte JDBC / ODBC foi recentemente adicionado. O protótipo inicial de HBase lançado em fevereiro 2007. O suporte para transações é atraente e incomum para um sistema de NoSQL.

4,2 HyperTable

HyperTable é escrito em C ++. Sua foi open-source por Zvents. Ele não parece ter decolado em popularidade ainda, mas Baidu tornou-se um patrocinador do projeto, que deve ajudar.

Hypertable é muito semelhante ao HBase e BigTable. Ele usa famílias de colunas que podem ter qualquer número de colunas "qualificadores". Ele usa timestamps de dados com MVCC.

Ele requer um sistema de arquivos distribuídos underyling como Hadoop, e um gerenciador de bloqueio distribuído. As tabelas são replicadas e repartiu-se sobre servidores por intervalos de chaves. As atualizações são feitas na memória e depois liberadas para o disco.

Hypertable suporta um número de interfaces de cliente de linguagem de programação. Ele usa uma linguagem de consulta chamado HQL.

4.3 Cassandra

Cassandra é semelhante às outras lojas de discos extensível em seu modelo de dados e funcionalidade básica. Ele tem grupos de colunas, as atualizações são armazenadas em cache na memória e depois descarregado para o disco ea representação disco é periodicamente compactado.

Ele faz o particionamento e replicação. detecção e recuperação de falhas são totalmente automáticas. No entanto, Cassandra tem um mais fraco modelo de concorrência do que alguns outros sistemas: não há nenhum mecanismo de bloqueio, e as réplicas são atualizados de forma assíncrona.

Como HBase, Cassandra é escrito em Java, e usadas sob licença Apache. É apoiado por DataStax, e foi originalmente código aberto pelo Facebook em 2008. Ele foi projetado por um engenheiro do Facebook e um engenheiro Dynamo, e é descrito como um casamento de Dynamo e BigTable. Cassandra é usado pelo Facebook, bem como outras empresas, de modo que o código é bastante madura. interfaces de cliente são criados usando a estrutura Thrift do Facebook:

http://incubator.apache.org/thrift/ Cassandra automaticamente traz novos nós disponíveis em um cluster, usa o algoritmo de competência phi para detectar falha de nó, e determina associação do cluster de forma distribuída com um algoritmo de estilo fofocas. Cassandra acrescenta o conceito de um "supercolumn" que fornece outro nível de agrupamento dentro de grupos de colunas. Bases de dados (chamados keyspaces) contêm famílias de colunas.

Uma família coluna contém tanto supercolumns ou colunas (não uma mistura de ambos). Supercolunns conter colunas. Como com os outros sistemas, qualquer linha pode ter qualquer combinação de valores da coluna (ou seja, as linhas são de comprimento variável e não são limitados por um esquema da tabela).

Cassandra usa um índice hash ordenado, que deve dar a maior parte do benefício de ambos os índices hash e B-árvore: você sabe que nós poderíamos ter um determinado intervalo de valores em vez de pesquisar todos os nós. No entanto, a classificação ainda seria mais lento do que com árvores-B.

Cassandra já teria escalado a cerca de 150 máquinas em produção no Facebook, talvez mais até agora. Cassandra parece estar ganhando muita força como um projeto open source, bem. Para aplicações onde Cassandra das eventual- modelo de consistência não é adequada "quorum lê" de uma maioria de réplicas fornecer uma maneira de obter os dados mais recentes. escreve Cassandra são atômicas dentro de uma família de coluna. Há também algum suporte para versões e resolução de conflitos.

4,4 outros Sistemas

sistema PNUTs do Yahoo também pertence à categoria "loja de discos extensível". No entanto, não é revisto neste documento, como é atualmente utilizado internamente para Yahoo. Nós também não analisou BigTable, embora a sua funcionalidade está disponível indiretamente por meio do Google Apps. Ambos os PNUTs e BigTable estão incluídos na tabela de comparação no final deste documento.

4.5 Resumo

As lojas de discos extensíveis são mais padronizados após BigTable. Eles são todos semelhantes, mas diferem em mecanismos de concorrência e outros recursos. Cassandra concentra-se em simultaneidade "fracos" (através MVCC) e HBase e HyperTable em consistência "forte" (por meio de fechaduras e o log).

5. RELATIONAL ESCALÁVEL SYSTEMS

Ao contrário dos outros armazenamentos de dados, SGBDs relacionais têm um esquema completo de pré-definido, uma interface SQL e transações ACID. Tradicionalmente, RDBMSs não alcançaram a escalabilidade do alguns dos precedentemente descrito armazenamentos de dados. A partir de 5 anos atrás, o MySQL Cluster apareceu a mais escalável, embora não seja muito alto desempenho por nó, em comparação com MySQL padrão.

Os desenvolvimentos recentes estão a mudar as coisas. Outras melhorias de desempenho foram feitas para MySQL Cluster, e vários novos produtos têm de sair, em particular VoltDB e Clustrix, que prometem ter boa por nó, assim como o desempenho escalabilidade. Parece provável que alguns relacional SGBDs irá fornecer escalabilidade comparável com armazenamentos de dados NoSQL, com duas ressalvas:

- *Use operações de pequena extensão:* Como vimos, as operações que se estendem por muitos nós, por exemplo, junta-se ao longo de muitos mesas, não escala bem com sharding.
- *Usar pequeno-escopo transações:* Da mesma forma, transações que abrangem muitos nós vai ser muito ineficiente, com a comunicação e de duas fases cometer sobrecarga. Note-se que os sistemas NoSQL evitar esses dois problemas, o que torna difícil ou impossível de realizar operações de escopo larger- e transações.

Em contraste, uma O RDBMS escaláveis não precisa *impedir* larger- operações de escopo e transações: eles simplesmente penalizar um cliente para essas operações *se eles usá-los.* RDBMSs escalável, portanto, têm uma vantagem sobre os armazenamentos de dados NoSQL, porque você tem a comodidade da linguagem SQL de nível superior e propriedades ACID, mas você só paga um preço para aqueles

quando eles abrangem nós. RDBMSs escaláveis são portanto, incluído como uma alternativa viável no presente documento.

5.1 MySQL Cluster

MySQL Cluster tem sido parte do lançamento do MySQL desde 2004, eo código evoluiu de um projeto ainda mais cedo da Ericsson. MySQL Cluster funciona substituindo o motor InnoDB com uma camada distribuída chamado NDB.

Ele está disponível a partir do MySQL (agora Oráculo); não é open source.

MySQL Cluster estilhaços de dados sobre vários servidores de banco de dados (um "compartilhada nada" arquitetura). Cada fragmento é replicada, para apoiar a recuperação. Bi-direcional replicação geográfica também é suportado. MySQL Cluster suporta em memória, bem como disk dados baseados.

Na memória de armazenamento permite em tempo real respostas.

Embora o MySQL Cluster parece ser dimensionado para mais nós do que outros RDBMSs até à data, que supostamente funciona em gargalos após algumas dezenas de nós. O trabalho continua no MySQL Cluster, de modo que este é susceptível de melhorar.

5.2 VoltDB

VoltDB é um novo RDBMS de código aberto projetado para alto desempenho (por nó), bem como escalabilidade. o escalabilidade e disponibilidade características está competitivo com MySQL Cluster e os sistemas NoSQL neste trabalho:

- Tabelas são divididos em vários servidores, e os clientes podem ligar para qualquer servidor. A distribuição é transparente para os usuários SQL, mas o cliente pode escolher o atributo sharding.
- Alternativamente, mesas seleccionados podem ser *replicado* sobre servidores, por exemplo, para acesso rápido a leitura principalmente dados.
- Em qualquer caso, cacos são replicados, para que os dados podem ser recuperados em caso de um acidente de nó. instantâneos de banco de dados também são suportados, contínua ou programada.

Algumas funcionalidades ainda estão desaparecidas, por exemplo, alterações de esquema online são actualmente limitada, e replicação assíncrona WAN e recuperação ainda não estão implementadas. No entanto, tem algumas características VoltDB promissores que colectivamente podem produzir uma ordem de magnitude vantagem no desempenho de um único nó. VoltDB elimina quase todos os "espera" na execução de SQL, permitindo uma implementação muito eficiente:

- O sistema é projetado para um banco de dados que se encaixa na RAM (distribuídos) nos servidores, de modo que o sistema nunca precisa esperar para o disco. Índices e estruturas de registo são concebidos para a RAM em vez de disco, e a sobrecarga de um cache de disco / tampão é eliminada bem. Desempenho será muito pobre se a memória virtual transborda RAM, mas o ganho com bom planejamento de capacidade RAM é substancial.
- execução de SQL é single-threaded para cada fragmento, usando uma arquitetura sem compartilhamento, então não há nenhuma sobrecarga para travamento multi-thread.
- Todas as chamadas SQL são feitas através de procedimentos armazenados, com cada procedimento armazenado ser uma transação. Isso significa, se os dados estão sharded para permitir transações para ser executado em um único nó, então não há necessidade de fechaduras, e, portanto, não espera em bloqueios. coordenação de transações é igualmente evitados.
- Os procedimentos armazenados são compilados para produzir código comparável às chamadas de nível de acesso de sistemas NoSQL. Eles podem ser executados na mesma ordem em um nó e no nó (s) réplica. VoltDB argumenta que essas otimizações reduzir muito o número de nós necessários para suportar uma determinada carga de aplicativos, com restrições modestos sobre o design de banco de dados. Eles já relataram alguns resultados de benchmark impressionantes em seu web site. Claro, o mais alto desempenho requer que o banco de dados trabalhando fits criados em RAM distribuída, talvez prorrogado por SSDs. Veja [5] por algum debate das questões arquitectónicas no VoltDB e sistemas similares.

5,3 Clustrix

Clustrix oferece um produto com semelhanças com VoltDB e MySQL Cluster, mas nós Clustrix são vendidos como aparelhos montados em rack. Eles afirmam escalabilidade para centenas de nós, com a fragmentação automática e replicação (com uma proporção de 4: 1 de leitura / gravação, eles relatam 350K TPS em 20 nós e 160M linhas). Failover é automático, e falhou nó recuperar é automático. Eles também usam discos de estado sólido para o desempenho adicional (como o MySQL Schooner e aparelhos NoSQL). Tal como acontece com outros produtos relacionais, Clustrix suporta SQL com operações totalmente ACID. distribuição de dados e balanceamento de carga é transparente para o programador da aplicação. Curiosamente, eles também projetou seu sistema para ser perfeitamente compatível com o MySQL, suportando aplicações MySQL existentes e conectores front-end.

5,4 ScaleDB

ScaleDB é um novo derivado de MySQL em curso. Como MySQL Cluster, ele substitui o motor InnoDB, e usa o agrupamento de vários servidores para alcançar escalabilidade. ScaleDB difere em que ele requer discos compartilhados entre os nós. Cada servidor deve ter acesso a cada disco. Esta arquitetura não tem escalado muito bem para Oracle RAC, no entanto.

sharding de ScaleDB é automático: mais servidores podem ser adicionados a qualquer momento. tratamento de falhas do servidor também é automática.

ScaleDB redistribui a carga sobre servidores existentes.

ScaleDB suporta transações ACID e bloqueio em nível de linha. Tem indexação multi-mesa (o que é possível devido ao disco compartilhado).

5.5 ScaleBase

ScaleBase leva uma nova abordagem, procurando atingir a escala horizontal com uma camada totalmente na parte superior do MySQL, em vez de modificar o MySQL. ScaleBase inclui um analisador SQL parcial e otimizador que os fragmentos tabelas ao longo de vários bancos de dados MySQL de nó único. Existe pouca informação sobre este novo sistema no momento da redação deste texto, no entanto. É atualmente uma versão beta de um produto comercial, não open source. Implementação sharding como uma camada em cima do MySQL introduz um problema, como transações não abrangem bancos de dados MySQL. ScaleBase fornece uma opção para a coordenação transação distribuída, mas a opção de desempenho superior que fornece transações ACID somente dentro de um único fragmento / servidor.

5.6 NimbusDB

NimbusDB é outro novo sistema relacional. Ele usa MVCC e armazenamento baseado em objectos distribuídos. SQL é a linguagem de acesso, com um otimizador de consulta orientada a linha e índices árvore AVL.

MVCC fornece isolamento de transação, sem a necessidade de bloqueios, permitindo o processamento paralelo em larga escala. Os dados são segmentados horizontalmente linha-por-linha em objectos distribuídos, permitindo multi-local, distribuição dinâmica.

5,7 outros Sistemas

Google criou recentemente uma camada sobre BigTable chamado Megastore. Megastore adiciona funcionalidade que traz BigTable mais perto de um SGBD relacionais (escaláveis) de muitas maneiras: transações que abrangem nós, um esquema de banco de dados definida em uma linguagem SQL-like, e caminhos hierárquicos que permitem que alguns limitada juntar capacidade. Google também implementou um processador de SQL que funciona em BigTable. Há ainda uma série de diferenças entre Megastore / BigTable "NoSQL" e sistemas relacionais escaláveis, mas a diferença parece estar diminuindo.

produto Azure da Microsoft tem algumas capacidades de replicação, mas não suporta diretamente escalar através de sharding. Ele permite que as tabelas sejam armazenados "na nuvem" e pode sincronizar vários bancos de dados. Ele suporta SQL. Nós não cobrimos-lo neste papel. A principal RDBMSs (DB2, Oracle, SQL Server) também incluem algumas características de escala horizontal, ou nada repartidos, ou de disco partilhado.

5.8 Resumo

MySQL Cluster usa uma arquitetura "compartilhada nada" para escalabilidade, como acontece com a maioria das outras soluções nesta seção, e é a solução mais madura aqui. VoltDB parece promissor devido à sua escala horizontal, bem como uma reformulação de baixo para cima para fornecer desempenho muito alto por nó. Clustrix parece promissor, bem como, e suporta discos de estado sólido, mas é baseado em software proprietário e hardware. Existe pouca informação sobre ScaleDB, NimbusDB e ScaleBase neste ponto; eles estão em um estágio inicial.

Em teoria, RDBMSs deve ser capaz de entregar escalabilidade enquanto aplicações evitar operações de cross-nó.

Se isso for verdade, na prática, a simplicidade de transações SQL e ácido lhes daria uma vantagem sobre NoSQL para a maioria das aplicações.

6. CASOS DE USO

Nenhum desses armazenamentos de dados é melhor para todos os usos. priorização de um usuário de recursos será diferente dependendo da aplicação, como será o tipo de escalabilidade necessária. Um guia completo para escolher um armazenamento de dados está além do escopo deste artigo, mas nesta seção veremos alguns exemplos de aplicações que se encaixam bem com as diferentes categorias de armazenamento de dados.

6.1 Valor-chave Exemplo loja

lojas de valor-chave são geralmente boas soluções se você tiver um aplicativo simples, com apenas um tipo de objeto, e você só precisa olhar para cima objetos se com base em um atributo. O simples funcionalidade de lojas de chave-valor pode torná-los mais simples de usar, especialmente se você já está familiarizado com memcached. Como exemplo, suponha que você tenha uma aplicação web que faz muitas RDBMS consultas para criar uma página personalizada quando um usuário faz login. Suponha que leva vários segundos para executar essas consultas, e os dados do usuário raramente é alterado, ou você sabe quando ele muda porque as atualizações passar pela mesma interface. Então você pode querer armazenar página personalizada do usuário como um único objeto em um armazenamento de chave-valor, representado de uma forma que é eficiente para enviar em resposta a solicitações do navegador e índice esses objetos por ID de usuário. Se você armazenar esses objetos persistentemente, então você pode ser capaz de evitar muitos RDBMS consultas, reconstruindo os objetos somente quando os dados de um usuário é atualizado. Mesmo no caso de um aplicativo como o Facebook, onde as mudanças da página inicial de um usuário com base nas atualizações feitas pelo usuário, bem como atualizações feitas por outros, pode ser possível executar RDBMS consultas apenas uma vez quando o usuário faz login, e para o resto do show de sessão apenas as alterações feitas por esse usuário (não por outra

Comercial). Em seguida, um armazenamento de chave-valor simples ainda pode ser usado como um cache de banco de dados relacional.

Você pode usar lojas de valor-chave para fazer pesquisas com base em vários atributos, através da criação de índices de valor-chave adicionais que você manter-se. No entanto, nesse ponto você provavelmente vai querer mudar para um armazenamento de documentos.

6.2 Loja documento Exemplo

Um bom exemplo de aplicação para uma loja de documento seria um com vários tipos diferentes de objetos (por exemplo, em um Departamento de aplicação Veículos Automotores, com veículos e motoristas), em que você precisa para procurar objetos com base em vários campos (digamos, um motorista de nome, número de licença, veículo de propriedade, ou data de nascimento). Um fator importante a considerar é o nível de concorrência garante que você precisa. Se você pode tolerar um modelo "eventualmente consistente" com atomicity limitada e isolamento, as lojas de documentos deve funcionar bem para você. Isso pode ser o caso na aplicação DMV, por exemplo, você não precisa saber se o motorista tem novas infrações de trânsito no passado minuto, e seria bastante improvável para dois escritórios DMV estar atualizando registro de motorista mesmo ao mesmo Tempo. Mas se você exigir que os dados ser up-to-date e atomicamente consistente, por exemplo,

6,3 Extensible Record Store Exemplo

Os casos de uso para lojas de discos extensíveis são semelhantes àquelas para armazenamentos de documentos: vários tipos de objetos, com pesquisas com base em qualquer campo. No entanto, os projectos loja de discos extensível são geralmente destinadas a uma maior

Taxa de transferência, e podem fornecer mais forte

garantias de concorrência, ao custo de um pouco mais complexidade do que as lojas de documentos.

Suponha que você está armazenando informações do cliente para uma aplicação eBay-style, e que deseja particionar seus dados horizontalmente e verticalmente:

- Você pode querer agrupar clientes por país, de modo que você pode eficientemente procurar todos os clientes em um país.

- Você pode querer separar o "core" informações de clientes raramente mudou, como endereços de clientes e endereços de email em um só lugar, e colocá

  certo freqüentemente atualizado cliente

  informações (como lances atuais em andamento) em um lugar diferente, para melhorar o desempenho. Embora você poderia fazer este tipo de horizontal / vertical dividindo-se em cima de uma loja de documento criando várias coleções para múltiplas dimensões, o particionamento é mais facilmente alcançado com uma loja de discos extensível como HBase ou HyperTable.

6,4 Exemplo RDBMS escalável

As vantagens da SGBD relacionais são bem conhecidos:

- Se a sua aplicação requer muitas tabelas com diferentes tipos de dados, uma definição de esquema relacional centraliza e simplifica a sua definição de dados e SQL simplifica enormemente a expressão de operações que abrangem tabelas.
- Muitos programadores já estão familiarizados com SQL, e muitos argumentam que o uso de SQL é mais simples do que os comandos de nível mais baixo fornecida por sistemas NoSQL.
- Transações simplificar codificação acesso simultâneo. semântica ácido livre o desenvolvedor de lidar com fechaduras, dados out-of-date, colisões de atualização e consistência.

- Muitos mais ferramentas estão atualmente disponíveis para SGBDs relacionais, para geração de relatórios, formulários e assim por diante.

Como um bom exemplo para relacional, imagine um aplicativo DMV mais complexo, talvez com uma interface de consulta para a aplicação da lei que pode interativamente pesquisar na cor do veículo, marca, modelo, ano, números de placas parciais, e / ou restrições sobre o proprietário, como do município de residência, cor do cabelo, e sexo. transações ACID também poderia ser valiosa para um banco de dados a ser atualizado a partir de vários locais, e as ferramentas acima mencionadas seria valioso bem. A definição de uma ferramentas comuns de esquema e administração relacionais também pode ser inestimável em um projeto com muitos programadores.

Estas vantagens são dependentes, é claro, de um SGBD relacional de escala para atender às suas necessidades de aplicação. benchmarks recentemente relatados na VoltDB, Clustrix, e a última versão do MySQL Cluster sugerem que a escalabilidade de SGBDs relacionais está melhorando muito. Mais uma vez, isso pressupõe que a sua aplicação não exige atualizações ou junta que se estendem por muitos nós; a coordenação de transações e dados de movimento para esse seria proibitivo. No entanto, os sistemas NoSQL geralmente não oferecem a possibilidade de transações ou consulta junta-se entre os nós, para que não são piores lá fora.

7. CONCLUSÕES

Nós cobrimos mais de vinte lojas de dados escaláveis neste trabalho. Quase todos eles são alvos móveis, com documentação limitada que é, por vezes conflitantes, de modo que este papel é provável out-of-date se já não estiver imprecisas no momento da redação deste texto. No entanto, vamos tentar um resumo instantâneo, comparação e previsões nesta seção. Considere isso um ponto de partida para um estudo mais aprofundado.

7.1 algumas previsões

Aqui estão algumas previsões do que vai acontecer com os sistemas que discutimos, ao longo dos próximos anos:

- Muitos desenvolvedores estarão dispostos a abandonar as transações globalmente ACID, a fim de ganhar escalabilidade, disponibilidade e outras vantagens. A popularidade dos sistemas NoSQL já demonstrou isso.

  Clientes tolerar companhia aérea sobre-reserva, e as ordens que são rejeitadas quando os itens em um carrinho de compras on-line são vendidos para fora antes do fim está finalizado. O mundo não é globalmente consistente.

- armazenamentos de dados NoSQL não será uma “moda passageira”. A simplicidade, flexibilidade e escalabilidade desses sistemas preenche um nicho de mercado, por exemplo, para sites com milhões de leitura / gravação usuários e esquemas de dados relativamente simples.

  Mesmo com melhorado escalabilidade relacional, sistemas NoSQL manter vantagens para algumas aplicações.

- Novas SGBDs relacionais também terá uma parte significativa do mercado de armazenamento de dados escalável. E se transações e consultas são geralmente limitados a nós únicos, estes sistemas devem ser capazes de dimensionar [5]. Quando o desejo de transações SQL ou ácido são importantes, estes sistemas será a escolha preferida.

- Muitas das lojas de dados escaláveis não vai provar “a empresa pronta” por um tempo. Mesmo que eles satisfazer uma necessidade, estes sistemas são novos e ainda não alcançaram a robustez, funcionalidade e maturidade de produtos de banco de dados que foram em torno de uma década ou mais. Early adopters já vimos quedas de web sites com falhas de armazenamento de dados escalável e muitos sites grandes continuar a “rolar sua própria solução” por sharding com produtos de RDBMS existentes. No entanto, alguns destes novos sistemas vão amadurecer rapidamente, dada a grande quantidade de energia dirigida a eles.

- Haverá maior consolidação entre os sistemas que descrevemos. Um ou dois sistemas provavelmente vai se tornar os líderes em cada uma das categorias. Parece improvável que o mercado e comunidade de código aberto será capaz de suportar o grande número de produtos e projetos que estudamos aqui. O capital de risco e apoio de jogadores-chave provavelmente será um fator nessa consolidação. Por exemplo, entre os armazenamentos de documentos, MongoDB recebeu investimento substancial neste ano.

7,2 SQL vs NoSQL

SQL (relacional) versus NoSQL escalabilidade é um tema controverso. Este artigo argumenta contra os dois extremos. Aqui está mais algumas informações para apoiar esta posição.

O argumento para relacional sobre NoSQL é algo como isto:

- Se novos sistemas relacionais pode fazer tudo que um sistema NoSQL podem, com desempenho semelhante e escalabilidade, e com a conveniência de transações e SQL, por que você escolher um sistema NoSQL?

- SGBDs relacionais têm tomado e mantido quota de mercado maioria sobre os outros concorrentes nos últimos 30 anos: rede, objeto e SGBDs XML.

- SGBDs relacionais bem sucedidos foram construídas para lidar com outras cargas de aplicação específica no passado: somente leitura ou leitura principalmente armazenamento de dados, OLTP em multi-core CPUs multi-disco, em memória bancos de dados, bases de dados repartidas, e agora horizontalmente dimensionado bases de dados.

- Enquanto não vemos “one size fits all” nos produtos SQL próprios, nós vemos uma interface comum com SQL, transações e esquema relacional que dá vantagens em treinamento, continuidade e intercâmbio de dados. O contra-argumento para NoSQL é algo como isto:

- Nós ainda não vimos bons benchmarks mostrando que RDBMSs pode alcançar escala comparável com os sistemas NoSQL como BigTable do Google.

- Se você apenas necessita de uma pesquisa de objetos com base em uma única chave, em seguida, um armazenamento de chave-valor é adequado e, provavelmente, mais fácil de entender do que um DBMS relacional. Da mesma forma para um armazenamento de documentos em uma aplicação simples: você só paga a curva de aprendizado para o nível de complexidade que você necessita.

- Algumas aplicações requerem um esquema flexível, permitindo que cada objeto em uma coleção para ter diferentes atributos. Enquanto alguns RDBMSs permitir “embalagem” eficiente de tuplas com falta atributos, e alguns permitem a adição de novos atributos em tempo de execução, isto é incomum.

- Um SGBD relacional torna as operações de “caro” (nó de multi-mesa multi) “muito fáceis”. sistemas NoSQL torná-los impossível ou, obviamente caro para programadores.

- Enquanto RDBMSs mantiveram participação de mercado da maioria sobre dos anos, outros produtos têm estabelecido mercados menores, mas não triviais em áreas onde há uma necessidade de capacidades particulares, por exemplo objetos indexados com produtos gostar BerkeleyDB, ou operações de seguimento de gráfico com SGBD orientada para objectos. Ambos os lados desse argumento tem mérito.

7.3 avaliação comparativa

Dado que a escalabilidade é o foco deste trabalho e dos sistemas que discutimos, há um "buraco" em nossa análise:

há uma escassez de benchmarks para fundamentar as muitas reivindicações feitas para escalabilidade. Como já observamos, há resultados de benchmark relatados em alguns dos sistemas, mas quase nenhum dos benchmarks são executados em mais de um sistema, e os resultados são geralmente relatado pelos defensores do que um sistema, por isso há sempre alguma pergunta sobre sua objetividade.

Neste artigo, nós tentamos fazer as melhores comparações possível com base em argumentos de arquitectura sozinho. No entanto, seria altamente desejável para obter alguns dados objetivos útil comparar as arquiteturas:

- Os trade-offs entre as arquiteturas não são claras. São os gargalos no acesso ao disco, comunicação de rede, As operações de índice, de bloqueio, ou outros componentes?
- Muitas pessoas gostariam de ver o apoio ou a refutação do argumento de que os novos sistemas relacionais pode escalar, bem como sistemas NoSQL.
- Um número de sistemas são novos, e pode não viver de acordo com reivindicações de escalabilidade sem anos de tuning. Eles também podem ser buggy. Que são verdadeiramente maduro?
- Quais sistemas com melhor desempenho no que carrega? São projetos de código aberto capaz de produzir sistemas com alto desempenho?

Talvez a melhor referência à data é de Yahoo! Investigação [2], comparando PNUTS, hbase, Cassandra, e MySQL sharded. Sua referência, YCSB, é projetado para ser representativa de aplicações web, eo código está disponível para os outros. Categoria 1 do medidas de benchmark de desempenho crus, mostrando características de latência como a carga do servidor aumenta. Nível de escala 2 medidas, mostrando como o sistema aferido escalas como servidores adicionais são adicionados, e quão rapidamente o sistema se adapta a servidores adicionais.

Neste artigo, eu gostaria de fazer uma "chamada para benchmarks de escalabilidade", sugerindo YCSB como uma boa base para a comparação. Mesmo se o ponto de referência YCSB é executado por diferentes grupos que não podem duplicar o mesmo hardware Yahoo especificado, os resultados serão informativo.

7,4 algumas comparações

Dada a rápida mudança da paisagem, este documento não tentará discutir os méritos de sistemas particulares, além dos comentários já feitos. No entanto, uma comparação das características mais salientes pode ser útil, por isso, terminar com algumas comparações.

A Tabela 1 abaixo compara o controle de concorrência, meio de armazenamento de dados, replicação, e transação mecanismos dos sistemas. Estas são difíceis de resumir em uma entrada da tabela curta, sem simplificação excessiva, mas nós comparamos as seguintes. Para a simultaneidade:

- Locks: alguns sistemas fornecem um mecanismo para permitir que apenas um usuário de cada vez para ler ou modificar uma entidade (um objeto, documento, ou linha). No caso de MongoDB, um mecanismo de bloqueio é fornecido com um nível de campo.
- MVCC: alguns sistemas fornecem controle de concorrência multi-versão, garantindo um ler-visão consistente do banco de dados, mas resultando em várias versões conflitantes de uma entidade se vários usuários modificá-lo ao mesmo tempo.
- Nenhum: alguns sistemas não fornecem atomicidade, permitindo que diferentes usuários para modificar diferentes partes do mesmo objeto em paralelo, e dando nenhuma garantia a respeito de qual versão de dados que você vai ter quando você ler.
- ACID: os sistemas relacionais fornecem transações ACID. Alguns dos sistemas mais recentes fazer isso sem bloqueios e sem esperas em serralharia, por operações de pré-análise para evitar conflitos. Para armazenamento de dados, alguns sistemas são projetados para armazenamento na RAM, talvez com instantâneos ou replicação para o disco, enquanto outros são projetados para armazenamento em disco, talvez cache na RAM.

sistemas baseados em RAM tipicamente permitir o uso de memória virtual do sistema operacional, mas o desempenho parece ser muito pobre quando transbordar RAM física. Alguns sistemas têm uma extremidade traseira conectável que permite meios de armazenamento de dados diferentes, ou eles exigem um padronizado sistema de arquivos subjacente.

A replicação pode garantir que espelham as cópias são sempre em sincronia (isto é, eles são atualizados lock-passo e uma operação não for concluída até que ambas as réplicas são modificados). Alternativamente, a cópia do espelho pode ser atualizado de forma assíncrona em a fundo. replicação assíncrona permite uma operação mais rápida, especialmente para réplicas remotas, mas algumas atualizações podem ser perdidos em um acidente. Alguns sistemas de atualizar cópias locais de forma síncrona e geograficamente cópias remotas de forma assíncrona (esta é provavelmente a única solução prática para dados remoto).

Transações são suportados em alguns sistemas, e não em outros. Alguns sistemas NoSQL fornecer algo entre os dois, onde as transações "locais" são suportados apenas dentro de um único objeto ou fragmento. A Tabela 1 compara os sistemas relativos a estes quatro dimensões.

Ta ble 1. Sistema C omparis o n (groupe d por categ o ry)

Sistema	conc contol	Dados Armazenamento Rep	ação	Tx
Redis	Locks	RAM Async		N
scalaris	Locks	RAM	Sincronizar	eu
Tóquio	Locks	RAM ou disco	Async	eu
Voldemort	MVCC RAM ou	BDB	Async	N
Riak	MVCC Plug-in Async			N
MemBrain	Locks	Flash + Disk	Sincronizar	eu
Membase	Locks	Disco	Sincronizar	eu
Dínamo	MVCC Plug-in Async			N
SimpleDB	Nenhum	S3	Async	N
MongoDB	Locks	Disco	Async	N
couch DB MVCC Disco			Async	N
Terrastore	Locks	RAM + Sincronizar		eu
HBase	Locks	Hadoop Async		eu
HyperTable Locks		arquivos	Sincronizar	eu
Cassandra	MVCC Disco		Async	eu
Mesa grande	Fechaduras + s Tamps	GFS	Sincronização + Async	eu
PNUTs	MVCC Disco		Async	eu
MySQL Cluster	ACID Disk		Sincronizar	Y
VoltDB	ÁCIDO, não feche RAM		Sincronizar	Y
Clustrix	ÁCIDO, não feche Disco		Sincronizar	Y
ScaleDB	ACID Disk		Sincronizar	Y
ScaleBase	ACID Disk		Async	Y
NimbusDB ACID,	não feche Disco		Sincronizar	Y

Outro fator a considerar, mas impossível de quantificar objetivamente em uma tabela, é a maturidade código. Como observado anteriormente, muitos dos sistemas que discutimos são apenas um par de anos, e são susceptíveis de ser confiável. Por esta razão, os produtos de banco de dados existentes são muitas vezes uma escolha melhor se eles podem escalar para as necessidades da sua aplicação.

Provavelmente o fator mais importante a considerar é o desempenho real e escalabilidade, como observado na discussão de benchmarking. referências de referência será estar adicionado para a website do autor cattell.net/datastores como eles se tornam disponíveis.

Atualizações e correções para este trabalho será postado lá também. A paisagem para armazenamentos de dados escaláveis é susceptível de alterar significativamente ao longo dos próximos dois anos!

8. AGRADECIMENTOS

Eu gostaria de agradecer Len Shapiro, Jonathan Ellis, Dan DeMaggio, Kyle Banker, John Busch, Darpan Dinker, David Van Couvering, Peter Zaitsev, Steve Yen, e Scott Jarr para a sua entrada em versões anteriores deste papel. Quaisquer erros são de minha autoria, no entanto! Eu também gostaria de agradecer a escuna Technologies por seu apoio neste papel.

9. REFERÊNCIAS

[1] F. Chang et ai, "BigTable: Um sistema de armazenamento distribuído para dados estruturados", *Sétimo Simpósio sobre design de sistema operacional e Implementação,* De Novembro de 2006. [2] B. Cooper et ai, "Aferição Nuvem Sistemas com YCSB Dose", *Simpósio ACM on Cloud Computing (* Socc), Indianapolis, Indiana, Junho De 2010.

[3] B. DeCandia et al, "Dynamo: da Amazon Altamente Disponível Key-Value Store", *Proceedings 21 st ACM Symposium SIGOPS sobre os Princípios sistemas operacionais,* De 2007.

[4] S. Gilbert e N. Lynch, "conjectura de Brewer e a viabilidade de serviços web consistentes, disponíveis, e partição tolerante", *ACM SIGACT Notícias 33,* 2, pp 51-59, Março de 2002.

[5] M. Stonebraker e R. Cattell, "Dez Regras para desempenho escalável em Datastores operação simples", *Comunicações da ACM,* Junho 2011.

10. Referências SISTEMA

A tabela a seguir fornece fontes de informação da web para todos os SGBDs e armazenamentos de dados abordados no papel, mesmo aqueles perifericamente mencionado, em ordem alfabética pelo nome do sistema. A tabela também lista o modelo de licenciamento (proprietário, Apache, BSD, GPL), que pode ser importante dependendo da sua aplicação.

Sistema	Site da licença para mais informações	
Berkeley DB BSD		oss.oracle.com/berkeley-db.html
Mesa grande	escorar	labs.google.com/papers/bigtable. html
Cassandra	Apache	incubator.apache.org/cassandra
Clustrix	escorar	clustrix.com
CouchDB	Apache	couchdb.apache.org
Dínamo	interno	portal.acm.org/citation.cfm?id=1 294281

GemFire	escorar	gemstone.com/products/gemfire
HBase	Apache	hbase.apache.org
HyperTable	GPL	hypertable.org
Membase	Apache	membase.com
MemBrain	escorar	schoonerinfotech.com/products/
memcached	BSD	memcached.org
MongoDB	GPL	mongodb.org
MySQL Cluster	GPL	mysql.com/cluster
NimbusDB	escorar	nimbusdb.com
Neo4j	AGPL	neo4j.org
OrientDB	Apache	orienttechnologies.com

PNUTs	interno	research.yahoo.com/node/2304
Redis	BSD	code.google.com/p/redis
Riak	Apache	riak.basho.com
scalaris	Apache	code.google.com/p/scalaris
ScaleBase	escorar	scalebase.com
ScaleDB	GPL	scaledb.com
SimpleDB	escorar	amazon.com/simpledb
Terrastore	Apache	code.google.com/terrastore
Tóquio	GPL	tokyocabinet.sourceforge.net
Versant	escorar	versant.com
Voldemort	Nenhum	project-voldemort.com
VoltDB	GPL	voltdb.com