Megastore: Solução para as crescentes exigências dos serviços na nuvem

Katharina Carrapatoso Garcia, Universidade Federal de São Carlos – campus Sorocaba

***Abstract* — O Megastore é um sistema de armazenamento desenvolvido por um time da empresa Google, para tentar suprir as necessidades dos serviços online disponíveis hoje na internet. Ele une duas tecnologias: o NoSQL, que garante escalabilidade, e sistemas de gerenciamento de Banco de Dados Relacional, que garantem maior facilidade e consistência. Com o Megastore é possível se ter alta disponibilidade e uma semântica ACID totalmente serializável em porções de fina granularidade de dados. Isso permite a replicação síncrona de operações, com baixa latência e recuperação de falhas eficiente.**

***Index Terms* — escalabilidade, NoSQL, Banco de dados relacional**

I. Introducão

O ACESSO à Internet hoje vem crescendo muito, e não só isso, mas o acesso à banda larga. Esse fato faz com que os usuários fiquem mais exigentes e os desenvolvedores criem diversas ferramentas e serviços diferenciados e interativos para suprir as necessidades e desejos dos usuários. Além disso, esses serviços estão migrando ou sendo desenvolvidos diretamente na nuvem, trazendo maior poder e capacidade, como redes sociais, serviços de email e desenvolvimento de documentos colaborativos, por exemplo.

Todos esses serviços exigem um alto poder de armazenamento e requisitos específicos. Um deles é manter a escalabilidade com milhões de usuários acessando simultaneamente um serviço todos os dias, o que é dificilmente alcançado com um banco de dados relacional, MySQL, por exemplo, pois esse não é o seu foco. Outro ponto é a competição natural que existe entre os serviços para conseguir usuários, isto é, não basta ser o primeiro a lançar uma ferramenta, deve-se ser o melhor e atingir o mercado com rapidez. Mais que isso, o serviço deve responder às requisições do usuário rapidamente, ou seja, ter baixa latência.

O serviço deve ainda se manter sempre consistente. Logo, se o usuário realiza uma alteração, ele não quer vê-la dali cinco minutos, ela deve ser instantânea, para todos os usuários. Por fim, a disponibilidade do serviço deve ser de 24 horas por dia, sete dias por semana.

Mas como conseguir isso se com as técnicas e tecnologias existentes hoje esses requisitos entram em conflito? Em banco de dados relacional, é possível se ter um desenvolvimento rápido e claro, porém não se tem escalabilidade. Já com NoSQL consegue-se alta escalabilidade, contudo se paga o preço da eventual falta de consistência e APIs limitadas.

O Megastore é um sistema de armazenamento de alta replicação que surgiu para suprir esses requisitos exigidos atualmente no mercado. Especialmente em relação ao serviço chamado *App Engine* da Google, que sofria de problemas sérios de confiabilidade[6].

O Megastore une o poder de escalabilidade do NoSQL[2] com a conveniência e garantia de consistência do banco de dados relacional, realizando replicações síncronas dos dados para garantir alta disponibilidade dos mesmos. Com isso, se mantém a semântica ACID sobre as replicas, com baixa latência servindo às ferramentas interativas.

A junção dessas duas abordagens é realizada separando-se os dados em pequenas partições, essas, então, são replicadas para vários datacenters separados geograficamente, sendo que cada uma destas replicas podem ser tratadas internamente utilizando-se toda semântica ACID, ou seja, mantendo sua consistência. Entretanto a consistência mantida entre as replicas é mínima.

Essa replicação que é realizada para os datacenters distribuídos em todas as operações de escrita é síncrona, e só é possível, pois foi utilizado o algoritmo Paxos[9] para garantir a baixa latência desejada.

O Megastore foi desenvolvido pela empresa Google, e já vem sendo estudado há alguns anos. Esse sistema roda sobre o Bigtable e consegue arcar com mais de três bilhões de operações de escrita e 20 bilhões de leituras por dia, além de armazenar aproximadamente um petabyte de dados primários.

Este artigo tem como objetivo apresentar as características e o funcionamento do Megastore, assim como explorar a sua estrutura. Na seção seguinte, será apresentado como é feita a replicação no sistema e o algoritmo de Paxos. Na terceira seção é apresentado a arquitetura do Megastore e em seguida na seção IV como é feito o particionamento. Na seção V como é realizado o tratamento de tabelas e a junção com o Bigtable da Google. Na seção VI são apresentados os algoritmos e estruturas de dados, e na seção seguinte, VII e VIII, uma análise de aplicações reais que utilizam o Megastore, e por fim a conclusão.

Artigo escrito em Maio 17, 2011.

C. G. Katharina, Universidade Federal de São Carlos, campus Sorocaba; e-mail: katharinacg@gmail.com

II. MEGASTORE

A. Replicação

O Megastore se utiliza de replicações síncronas espalhadas em datacenters distribuídos em distantes áreas geográficas, garantindo assim a disponibilidade. Desta forma, caso ocorra algum evento em um datacenter, como incêndio, ou queda, ou qualquer outro incidente que ameace a perda dos dados, eles estarão disponíveis em algum outro local.

Para isso o time desenvolvedor do Megastore se concentrou em algumas estratégias já consolidadas de replicação, se focando em não considerar aquelas que aceitam perda de dados, caso ocorra falha. Também foram desconsideradas estratégias que não permitissem a realização de transações ACID, afinal isso quebraria todo propósito do Megastore. As estratégias analisadas foram as seguintes:

1) Mestre/Escravo Assíncrono:
Um nó mestre faz a replicação de um log de escritas para um escravo, pelo menos. Adições a serem feitas no log são reconhecidas pelo mestre, e transmitidas aos escravos. Com isso o mestre consegue realizar transações ACID rápidas, porém pode se ter lentidão ou perda de dados caso se ocorra a falha de um escravo.

2) Mestre/Escravo Síncrono:
Um mestre aguarda as mudanças serem replicadas para todos os escravos antes de confirmá-las, evitando perda de dados em caso de falhas.

3) Replicação otimista:
Qualquer membro de um grupo de replicas pode aceitar alterações, que são propagadas assincronamente pelo grupo. Com isso, se tem ótima disponibilidade e baixa latência, porém transações não podem ser realizadas, pois não há como saber a ordem em que as mudanças ocorreram em cada replica.

Foram evitadas também estratégias que tivessem um mestre sobrecarregado, para evitar tratamentos de falhas pesados e de alta complexidade. Dessa forma, não é necessário ter um mestre distinto dos escravos. Para se conseguir isso foi utilizado o algoritmo de Paxos, visto que estratégias comuns não condizem com o objetivo do Megastore.

B. Paxos

O algoritmo de Paxos foi escolhido para realizar uma replicação otimizada, conseguindo-se alcançar um consenso no grupo de replicas para ficar com um único valor consistente. Essa abordagem ganhou espaço no projeto Megastore, pois o Paxos é um algoritmo em que não há necessidade de se designar um servidor mestre, e porque é ótimo e possui grande tolerância a falhas.

O algoritmo é aplicado da seguinte forma: Um log é replicado em um grupo de *peers* simétricos, em que qualquer um pode iniciar operações de leitura e escrita. A maioria dos *peers* deve estar ativo para que o algoritmo funcione corretamente. A maioria "escolhe" o valor que será mantido, e a minoria, que está inconsistente, irá se adaptar em seguida; O log gerado possui várias instancias de Paxos. Cada instancia corresponde a uma alteração que será adicionada, e possui uma posição fixa no log, que seria abaixo da última alteração feita.

Como o algoritmo é tolerante a falhas, não é necessário se criar um estado que indique que ocorreu falha. Porém o algoritmo em sua estrutura original não cobria todos os requisitos que o Megastore precisava, por esse motivo, foram feitas alguma alterações, as quais permitem leituras rápidas, escritas rápidas e tipos de replicas.

As leituras rápidas podem ser realizadas, pois como escritas usualmente são executadas com sucesso, é possível se ter leituras locais, em uma replica. Para garantir que essa leitura local pode ser feita, foi desenvolvido o *Coordenador* presente no datacenter de cada replica. Ele controla se a replica em questão está atualizada com todas as escritas realizadas pelo Paxos no log, e, em caso afirmativo, permite que seja feita a leitura local.

As escritas rápidas são atingidas se usando *líderes* ao invés de *mestres* para controlar as operações de escrita. A diferença é que existe um líder para cada nova posição a ser alterada no log, ou seja, para cada instancia do Paxos. O líder é geralmente o último a alterar os dados, pois é comum que se tenha uma sequência de operações vindas de um mesmo ponto. O líder controla o valor que será aceito para ser escrito, então aquele que realizar a escrita primeiramente, irá submeter o novo valor ao líder e verificar se as outras replicas aceitam a alteração.

Os tipos de replicas disponíveis são três: *full*, *witness* e *read-only*.

O tipo full consiste das replicas completas que permitem que sejam realizadas leituras atualizadas. Já o tipo witness participa da "votação" realizada para definir o que será adicionado no log. Este tipo armazena o log, porém não atualiza os dados de suas entidades, ou seja, elas possuem baixo custo de armazenamento. O witness é usado para resolver empates na votação, e ainda quando não existem replicas suficientes para formar a maioria necessária.

Por fim as replicas do tipo read-only são o inverso das replicas do tipo witness. Elas contém *snapshots* completos dos dados. Leituras dessas replicas refletem um estado consistente dos dados que não é o corrente, mas não é muito antigo, o que permite disseminar os dados sem impactar na latência, quando um pequeno "atraso" dos dados é aceitável.

Adicionado a isso, existe a preocupação de que, com replicas espalhadas por diferentes datacenters em áreas extensas, a utilização de um único log poderia causar um aumento no *throughput*. Portanto, são utilizados vários logs replicados, cada um com uma parte de um conjunto de dados.

III. ARQUITETURA

A. Lógica

O Megastore é executado por servidores auxiliares e uma biblioteca cliente, a qual implementa o Paxos. Isso permite que as aplicações se conectem com a biblioteca cliente para utilizar o sistema.

Cada servidor de aplicação possui uma replica local designada. Para tornar as operações realizadas em Paxos permanentes são submetidas transações diretamente na Bigtable local. Para minimizar as transações entre grandes distâncias físicas, a biblioteca submete operações remotas do Paxos para servidores de replicação temporários que se comunicam com a Bigtable local. Esses servidores são úteis também, pois eles buscam por operações de escrita incompletas e tentam transformá-las em completas propondo valores no-op (nenhuma operação realizada) via Paxos.

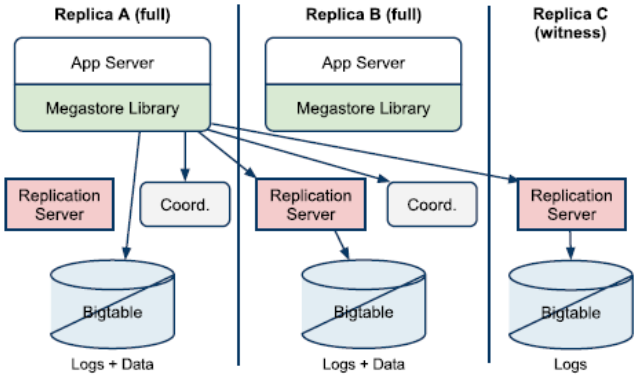

Figura 1 - Exemplo de arquitetura do Megastore

A Figura 1 apresenta um exemplo da arquitetura do Megastore, para uma instancia com 2 replicas do tipo full e uma do tipo witness. Nesse exemplo é possível perceber claramente a diferença dos tipos de replica.

B. *Física*

Como base NoSQL é utilizado o Bigtable da Google, garantindo a escalabilidade e tolerância a falhas, pois as operações são espalhadas pelas múltiplas colunas. Para diminuir a latência é possível escolher a instancia de Bigtable que será usada, e a localização. Ou seja, as aplicações tendem a utilizar instancias próximas de seus usuários finais, e replicas próximas umas das outras. Apenas algumas replicas são armazenadas isoladas para proteção.

Dentro do Bigtable é possível armazenar dados mais acessados e relacionados em colunas mais próximas, ou desnormalizados em uma mesma coluna.

IV. Particionamento

O Megastore permite que as aplicações particionem seus dados com alta granularidade garantindo escalabilidade, e um bom desempenho.

A. *Entity Groups*

Os dados no Megastore são particionados em coleções chamadas de *entity groups.* Cada um desses grupos é replicado independentemente e sincronicamente sobre uma grande área. Como já mencionado, esses dados são armazenados em um banco de dados NoSQL, no caso o Bigtable, em cada datacenter.

Internamente a cada grupo, as entidades podem ser manipuladas com transações ACID, as quais são replicadas utilizando Paxos. Já a comunicação realizada entre os *entity groups* é realizada via mensagens assíncronas. Logo, os grupos recebem as mensagens, que vão sendo armazenadas em uma fila. Essas mensagens vão sendo consumidas e as alterações de escrita sendo aplicadas.

Entretanto, para que a estratégia do Megastore seja eficaz, é necessário ter um bom limiar para o particionamento dos dados. Precisa-se definir o quanto uma partição de dados será granularisada, ou seja, como será feita a divisão dos dados em pequenos grupos. Isso é importante, pois caso se tenha grupos excessivamente pequenos se terá a consequência de uma grande quantidade de operações entre grupos que terão que ser realizadas. Por outro lado, se os grupos tiverem dados excessivos e não relacionados, se terá muitas escritas o que degrada o *throughput*.

Existem alguns exemplos que podem ser considerados:

1) Email: Cada conta de email acarreta em um *entity group*. Isso porque as operações realizadas dentro da conta de um usuário devem ser transacionais e consistentes. Ou seja, o usuário deve conseguir visualizaras mudanças em sua conta, como marcação de um email ou envio, mesmo que ocorra uma falha em alguma replica.

2) Blog: Cada usuário, assim como na conta de email, possui uma *entity group*. Em relação ao conteúdo do blog, se mantem um *entity group* para armazenar as postagens e metadados para cada blog. Isso porque os blogs são ferramentas colaborativas e seu conteúdo não está atrelado somente ao seu proprietário, mesmo por que esse pode se modificar. Outro grupo abrange a chave para unificar o nome escolhido para cada blog.

3) Maps: Como não existe um padrão conveniente para ser utilizado para se particionar dados geográficos, já que cada local do mundo possui diferentes conceitos em relação a endereços e cidades e diferentes características são criados caminhos (patches) que dividem o mundo e que não se sobrepõe. Cada um desses patches gera uma entity group. Eles devem ser suficientemente grandes para que não sejam necessárias muitas alterações no patch em si, pois elas teriam que ser feitas com *two-phase commit[7]*, para a mudança ser atômica. Mas não tão grandes para que não tenham que ser realizadas operações de escrita que denigrem o throughput.

V. Modelo de dados

O modelo de dados do Megastore se encontra entra tuplas Banco de Dados Relacional e as colunas do armazenamento do NoSQL. Existe então a declaração de um *schema* que é fortemente tipado. O *schema* possui tabelas, em que cada uma possui um conjunto de entidades, que possuem suas devidas propriedades. As quais podem assumir valores de strings, valores numéricos e Buffers do protocolo da Google. Todas as entidades de uma tabela possuem o mesmo conjunto de

propriedades, as quais formam a *primary key* dessa entidade.

Existem dois tipos de tabela no Megastore:

1) *Entity group root:* Um tabela base que não possui *foreign key*.
2) *Child:* Cada uma dessas tabelas referencia uma tabela *root* via *foreign key*.

Todo *entity group* possui uma *entity group root* e várias *childs* que referenciam essa *root.* O exemplo que segue na Figura 2 demonstra uma estrutura de uma aplicação de usuários e sua coleção de fotos.

```
CREATE SCHEMA PhotoApp;

CREATE TABLE User {
 required int64 user_id;
 required string name;
} PRIMARY KEY(user_id), ENTITY GROUP ROOT;

CREATE TABLE Photo {
 required int64 user_id;
 required int32 photo_id;
 required int64 time;
 required string full_url;
 optional string thumbnail_url;
 repeated string tag;
} PRIMARY KEY(user_id, photo_id),
  IN TABLE User,
  ENTITY GROUP KEY(user_id) REFERENCES User;

CREATE LOCAL INDEX PhotosByTime
  ON Photo(user_id, time);

CREATE GLOBAL INDEX PhotosByTag
  ON Photo(tag) STORING (thumbnail_url);
```

Figura 2 - Exemplo de um schema no Megastore

Esse exemplo apresenta um *schema* chamado PhotoApp que possui as entidades User, que é o *root* e a entidade Photo que é uma *child* de User.

B. *Chaves*

As chaves (keys) do modelo do Megastore não são utilizadas como em um banco de dados relacional comum, onde elas assumem valores surrogate[8]. No Megastore as chaves são usadas para agrupar entidades que serão lidas no mesmo momento. Cada entidade é mapeada para uma linha da Bigtable, e a chave dessa coluna é a primary key da entidade, e suas outras propriedades possuem uma coluna cada.

Como na Figura 2 temos as tabelas User e Photo com a mesma chave user_id, que é na tabela Photo uma foreign key, elas serão alocadas no mesmo Bigtable, e que as entidades Photo serão colocadas adjacentes a seus usuários correspondentes. Com isso é possível se gerar um layout hierárquico manipulando essas chaves.

C. *Índices*

Existem dois tipos de índices no Megastore:

1) *Índice Local:* um índice local é um índice único para cada entity group. Ele pode ser utilizado para se selecionar dados dentro de um entity group. Um exemplo é o PhotosByTime encontrado na Figura 2. Os novos ídices são armazenados atomicamente e em conformidade com os dados primários da entidade.
2) *Índice Global:* o índice global é utilizado para se encontrar entidades sem saber inicialmente a qual entity group elas pertencem. Um exemplo é o PhotosByTag presente no exemplo da Figura 2. Ele permite que fotos sejam encontradas e marcadas com uma tag, independentemente de quem postou a foto.

Algumas outras funcionalidades para índices são apresentadas no exemplo da Figura 2. Primeiramente, podemos ver uma cláusula STORING no índice global PhotosByTag. Isso faz com que seja possível armazenar a Url do thumbnail da foto no índice, para uma recuperação mais rápida.

Depois pode-se notar a cláusula repeated na definição do índice PhotosByTag na tabela Photo. Essa definição faz com que toda vez que uma nova entrada de tag é colocada na tabela Photo, um novo index é criado.

D. *Mapeamento para Bigtable*

Cada coluna no Bigtable tem como nome, o nome da tabela mais o nome da propriedade que será representada por aquela coluna. Desta forma, várias entidades podem estar na mesma Bigtable sem que haja colisão.

Na Figura 3 é possível ver o mapeamento do exemplo apresentado na Figura 2. Para a tabela root do schema são armazenados além das propriedades, o log de transações com os metadados das replicações e transações. Desta forma, como todos esses dados estão em apenas uma linha, é possível atualizar tudo com apenas uma transação em Bigtable.

Row key	User. name	Photo. time	Photo. tag	Photo. _url
101	John			
101,500		12:30:01	Dinner, Paris	...
101,502		12:15:22	Betty, Paris	...
102	Mary			

Figura 3 – Exemplo mapemento para Bigtable

VI. ESTRUTURAS DE DADOS E ALGORITMOS

A. *Logs*

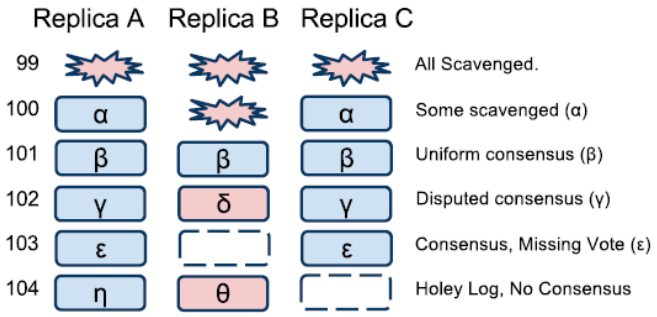

Figura 4 - Exemplo de logs em replicas de um entity group

No Megastore é permitido que uma replica tenha "buracos" em relação às outras.

No exemplo da Figura 4 é possível ver esse cenário em um entity group. Como já explicado, existe uma votação para que se escolha o valor que será colocado no log por Paxos. Pelo exemplo é possível notar as diferentes situações que podem ocorrer. Na posição 101 do log, todas as replicas aceitaram a operação. Na posição 103 a replica B não consegue votar, mas A e C concordam. Na posição 104 a replica C não possui voto, e A e B não concordam e a posição permanece com um buraco, pois ocorreu um conflito e a operação não pode ser adicionada ao log ou se teria inconsistência.

B. *Leitura*

Existe a necessidade de se realizar leituras correntes, isto é, uma leitura em que todos os dados estejam atualizados. Para isso, é preciso que ao menos uma replica esteja 100% completa e com todas as transações finalizadas, isso se chama realizar o catchup.

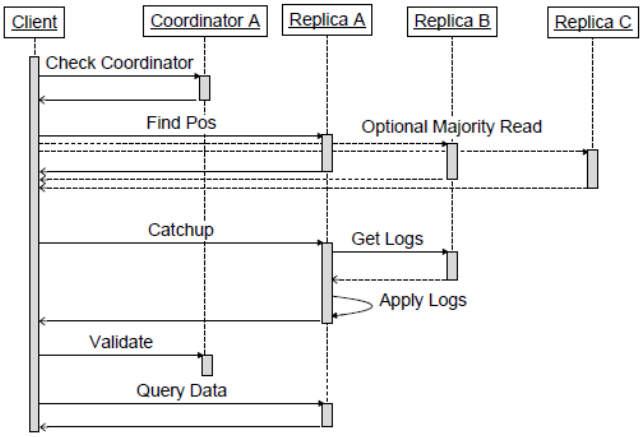

Figura 5 - Leitura corrente

Como é possível ver na Figura 5, são necessários cinco passos para se conseguir fazer a leitura corrente:

1) verificar com o coordinator se o entity group está atualizado localmente.

2) verificar a posição mais atualizada e consistente do log. Se o coordinator indicar que a replica local esta atualizada, ler dela mesma. Se não outra replica deve ser escolhida que esteja o mais atualizada possível.

3) Catchup: A replica escolhida deve se atualizar para a ultima posição do log. Se há alguma posição em que a replica escolhida tenha ficado com um buraco, o valor deve ser lido de outra replica. Para as posições que ficaram em conflito, Paxos é chamado para verificar com a maioria das replicas se deve permanecer um valor no-op ou um dos valores indicados anteriormente. Em qualquer ocorrência de falha, outra replica será escolhida.

4) A replica então é validada pelo coordinator como estando atualizada.

5) A leitura é finalmente realizada. Em caso de falha o catchup é realizado novamente.

C. *Escrita*

O algoritmo de escrita é composto também de 5 passos, como pode-se ver na Figura 6:

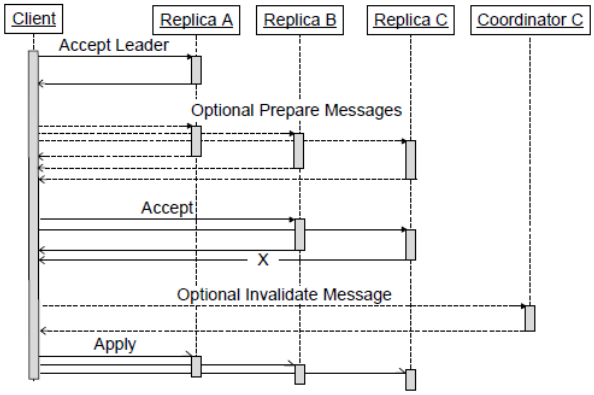

Figura 6 – Escrita

1) Após o final da leitura corrente uma replica foi colocada como novo líder. Esse líder irá receber a nova operação de escrita que quer ser realizada. Se o líder aceitar segue-se para o passo 3.

2) A fase de preparação do algoritmo de Paxos será executada, na qual são analisadas todas as replicas com possíveis valores para aquela posição do log que está sendo tratada. O valor mais encontrado irá substituir o valor proposto

3) Todas as replicas são requisitadas para aceitar o novo valor. Se a maioria falhar, o passo dois deve ser executado.

4) Para as replicas que não aceitaram, o coordinator deve ser invalidado, pois suas replicas não estão mais atualizadas.

5) Executar as alterações nos dados efetivamente.

Executar o passo cinco apenas após todos os logs serem atualizados e os coordinators, garante que a consistência não será violada, pois se sabe qual replica pode efetuar a escrita e qual não.

VII. Tratamento de falhas

Existe a possibilidade de que uma ou mais replicas full venham a falhar, e quando isso ocorre a performance do Megastore pode diminuir. Para tratar essa situação a primeira tarefa a ser feita é desviar o tráfego da aplicação que estava indo para aquela replica para outras replicas. Em seguida é necessário desabilitar o coordinator, para que ele não influencie nas outras escritas. Mas como a replica não é totalmente desabilitada ela ainda participa no primeiro passo do algoritmo de escrita.

Uma ação raramente usada é desabilitar totalmente a replica. Apesar de parecer simples, isso atinge a disponibilidade, pois se tem uma replica a menos. Logo, isso só é feito em casos onde a replica não desabilitada está sobrecarregando o sistema.

VIII. Avaliação

Existem mais de cem aplicações utilizando o sistema do Megastore como armazenamento. As figuras 7 e 8 apresentam a avaliação de disponibilidade de cada aplicação e latência de leituras e escritas respectivamente. É possível perceber que a maioria das aplicações obtém alta porcentagem de sucesso em suas operações. Porém a latência deve ser analisada com cuidado, pois ela depende muito de como os dados estão particionados.

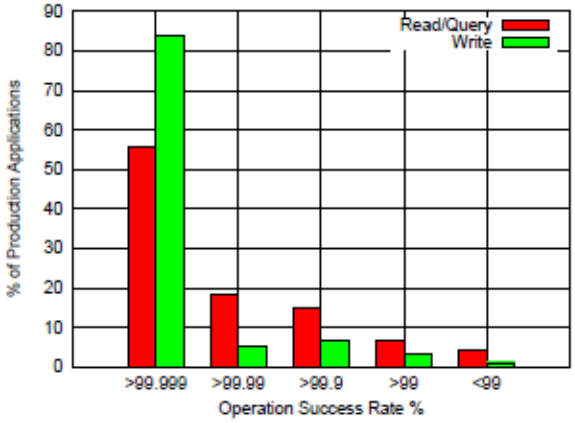

Figura 7 - Análise da disponibilidade

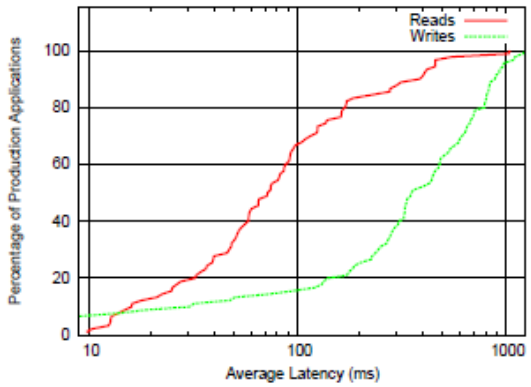

Figura 8 - Análise da latência

Como o Megastore utiliza 3 vezes o número de recursos do que um armazenamento comum Mestre/Escravo ele custa 3 vezes mais. E sendo um banco de dados distribuído, considerando o teorema de CAP [2], a aplicação se torna mais complexa e custosa. Por esse motivo o Megastore é direcionado a aplicações criticas, que necessitam de replicas espalhadas.

IX. Conclusão

Neste artigo foi apresentado o sistema de armazenamento Megastore, que une o NoSQL ao banco de dados relacional, proporcionando uma solução para o problema de crescente demanda de armazenamento, escalabilidade e disponibilidade que existe hoje na Internet. A aproximação explorada pelo Megastore apresenta o que pode ser uma tendência do mercado atual, onde se tenta alcançar todos os requisitos possíveis mesmo que minimamente em alguns momentos. Como acontece no Megastore na comunicação entre replicas, onde a consistência é minimamente mantida, porém se consegue alta escalabilidade, e a consistência é altamente preservada dentro de cada replica, já que se consegue realizar transações ACID fácil e rapidamente.

Existem hoje mais de 100 aplicações que já utilizam o Megastore como sistema de armazenamento. Isso demonstra que a abordagem apresentada está começando a ser aceita e pode ser usada em aplicações reais.

Referências

[1] J. Baker, C. Bond, J. C. Corbett, JJ Furman, A. Khorlin, J. Larson, J-M. Léon, Y. Li, A. Lloyd, V. Yushprakh – Google Inc., “Megastore: Providing Scalable, Highly Available Storage for Interactive Services” in *CIDR 2011*, Asilomar, California, USA: Janeiro 9-12, 2011.

[2] R. M. Toth, “Abordagem NoSQL – Uma real alternative” Sorocaba, São Paulo, Brasil: Abril 13, 2011.

[3] P. Bernstein, “Google Megastore” in http://perspectives.mvdirona.com Julho 10, 2008.

[4] J. Hamilton, “Google Megastore: The Data Engine behind GAE” in http://perspectives.mvdirona.com Janeiro 9, 2011.

[5] T. Hoff, “Google Megastore - 3 Billion Writes And 20 Billion Read Transactions Daily” in http://highscalability.com Janeiro 11, 2011

[6] K. Finley, “Google Announces High Replication Datastore for App Engine” in www.readwriteweb.com Janeiro 6, 2011

[7] A. Girbal, “Two-phase commit” in www.mongodb.org Março 14, 2011

[8] C. Lima, “Conceito de Surrogate Key – Chaves Substitutas” Março 9,2011

[9] L. Lamport, “Paxos made simple” Novembro 1, 2011