Spanner

Google's Globally-Distributed Database

(Banco de Dados Globalmente Distribuído da Google)

James C. Cobertt, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, et. al.

Google, Inc.

Universidade Federal do Paraná - UFPR

Programa de Pós-Graduação em Informática - PPGInf

Oficina de Banco de Dados - CI829

- **Professora: Dra. Carmem Hara**
- **Aluno: Walmir Couto**

Curitiba-PR 15/10/2013

Apresentação

- Resumo
- Introdução
- Implementação
  - Spanserver Software Stack
  - Diretórios e Localização
  - Modelo de Dados
- TrueTime API
- Avaliação
- Trabalhos Relacionados
- Trabalhos Futuros
- Conclusões

Resumo

- Spanner é banco de dados escalável, multi-versão, globalmente distribuído e replicado sincronamente da Google;
- É o primeiro sistema a distribuir dados em escala global e a suportar transações distribuídas externamente consistentes;
- O artigo descreve como o Spanner está estruturado, seus recursos, as decisões de projeto lógico-relacional, e uma nova API que expõe a suspensão de *clock*;
- Esta API e sua implementação são pontos críticos para o suporte à consistência externa e a uma variedade de recursos poderosos:
    - Leituras sem-*block*;
    - Transações somente-leitura *lock-free*;
    - Mudança de schema atômico.

Introdução

- O Spanner é um BD que fragmenta dados sobre vários conjuntos de máquinas de estado replicadas Paxos em *datacenters* espalhados ao redor do mundo. Paxos é uma família de protocolos para a solução de consenso em uma rede de processadores não confiáveis;

- O Spanner refragmenta, replica e migra (automaticamente) dados através das máquinas (ou mesmo através dos *datacenters*) para balancear a carga;

- O Spanner é projetado para escalabilidade de milhares de máquinas, através de centenas de *datacenters* e trilhões de linhas de BD;

- Aplicações podem usar o Spanner para obter alta disponiblidade, mesmo diante de desastres de grandes proporções, pela replicação de seus dados (dentro ou através de continentes);

Introdução

- O cliente inicial do Spanner foi o F1: SGBD Relacional Distribuído Tolerante à Falha que dá suporte aos negócios de publicidade da Google;

- O F1 usa 5 réplicas espalhadas pelos EUA;

- Outras aplicações irão replicar seus dados através de 3 a 5 *datacenters* em uma determinada região geográfica, mas com modos de falha relativamente independentes;

- Isto é, muitas aplicações optam pela baixa latência à alta disponibilidade, bem como elas podem sobreviver a 1 ou 2 *datacenters* falhos;

Introdução

- O foco principal do Spanner está na gestão dos dados replicados através dos *datacenters*;
- Muitos projetos usam o Bigtable, porém a Google tem recebido diversas mensagens de usários com reclamações de que o Bigtable pode ser difícil de usar para alguns tipos de aplicações: complexidade de *schemas*, necessidade de consistência forte na presença de replicações
- Muitas aplicações na Google usam o Megastore por causa do seu modelo de dados semi-relacional e suporte para replicações síncronas (apesar do Megastore ser relativamente pobre em relação à vazão de escrita);

Introdução

- O Spanner envolve uma versão de armazenamento chave-valor (como no Bigtable) dentro de um BD multi-versão temporal;

- O dado é armazenado em tabela semi-relacional *"schematizada"*;

- Cada dado ganha sua versão, e cada versão é automaticamente *"timestamped"* com o seu *commit time*;

- O Spanner suporta transações de propósito geral, e fornece uma linguagem de consulta baseada em SQL;

Introdução

- Por ser um BD distribuído globalmente, o Spanner fornece vários recursos interessantes:
  - ✓ As configurações de replicações podem ser controladas dinamicamente;
  - ✓ Dados podem ser movidos dinamicamente e transparentemente entre os *datacenters* para balancear o uso dos recursos;
- O Spanner atribui *timestamp* de *commit* global para as transações (mesmo que as transações possam ser distribuídas);
- O *timestamp* reflete a ordem de serialização;

Introdução

- A ordem de serialização satisfaz a consistência externa:
  - Se a transação T_1 faz o *commit* antes da transação T_2 começar...
    - então o *timestamp* de *commit* de T_1 é menor do que o de T_2;
- O Spanner é o primeiro sistema a fornecer tais garantias em escala global;
- O segredo está na nova TrueTime API e na sua implementação;

Introdução

- API expõe diretamente as suspensões de *clock*, e as garantias nos *timestamps* do Spanner dependem dos limites que a implementação fornece;
- Se a suspensão é grande, o Spanner diminui para esperar aquela suspensão;
- O software de gerenciamento de *clusters* da Google fornece uma implementação da TrueTime API;
- Esta implementação mantém pequena suspensão (geralmente menor do que 10ms) pelo uso de múltiplas referências de *clocks* modernos (GPS e *atomic clocks*).

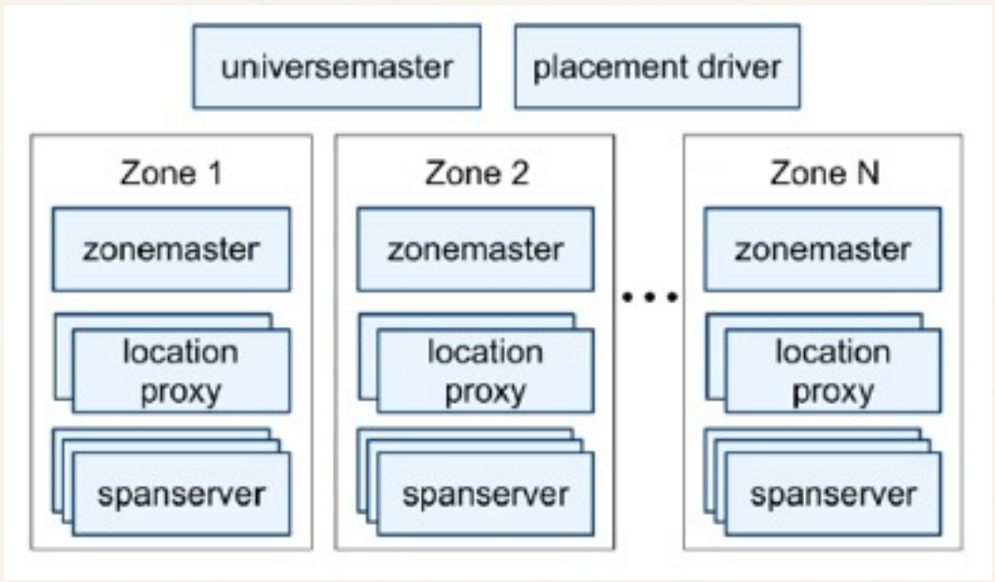
Implementação

- Veremos o porquê do Spanner se parecer com um BD relacional ao invés de um armazenamento chave-valor, e como as aplicações podem controlar a localidade dos dados;
- Uma implementação do Spanner é chamada de universo;
- Como o Spanner gerencia dados globalmente, haverá diversos universos em execução (normalmente):
  - ✓ universo test/playground;
  - ✓ universo desenvolvimento/produção;
  - ✓ universo somente produção;

Implementação

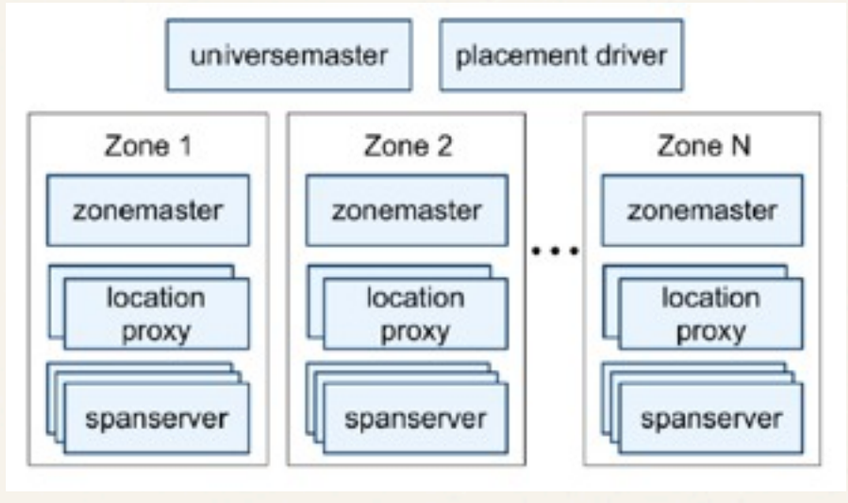
- O Spanner está organizado como um conjunto de zonas, onde cada zona é (a grosso modo) um servidor Bigtable;
- Zonas são as unidades de desenvolvimento administrativo;
- O conj. de zonas é também o conj. de localidades onde os dados podem ser replicados;
- Zonas podem ser adicionadas e/ou removidas a partir de um sistema em execução (assim como novos *datacenters* são postos em serviço e os antigos são desativados);
- Zonas também são as unidades de isolamento físico: podem existir uma ou mais zonas em um *datacenter*;
- Ex: se dados de aplicações diferentes devem ser particionados através de diferentes conjuntos no mesmo datacenter;

Implementação



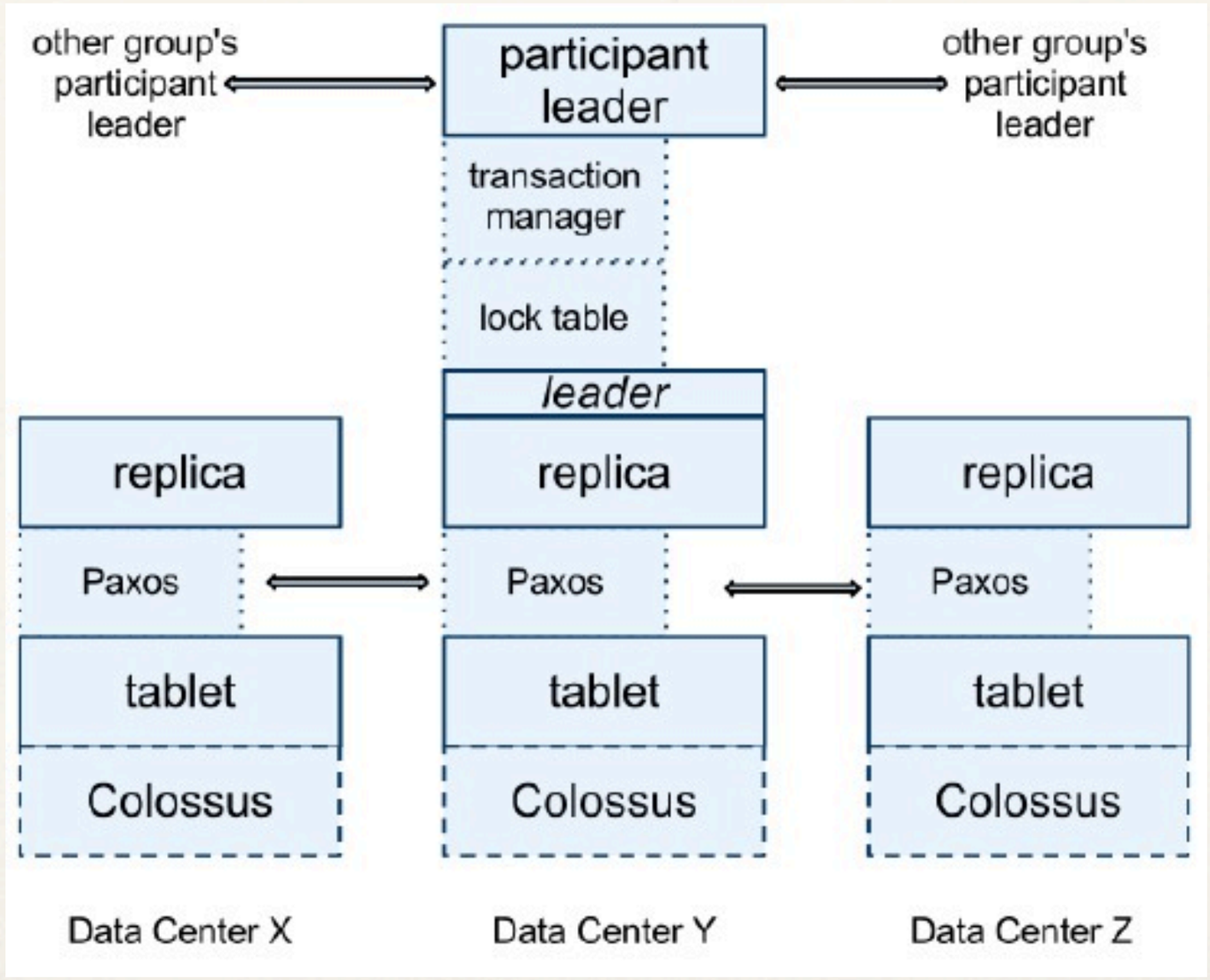
- A figura ilustra os servidores em um universo Spanner;
- Uma zona tem um *zonemaster* e entre uma centena e vários milhares de *spanservers*;
- O *zonemaster* atribui dados aos *spanservers*, e este serve dados aos clientes;
- Os *location proxies* por zona são usados pelos clientes para localizar os *spanservers* designados a servir seus dados;

Implementação

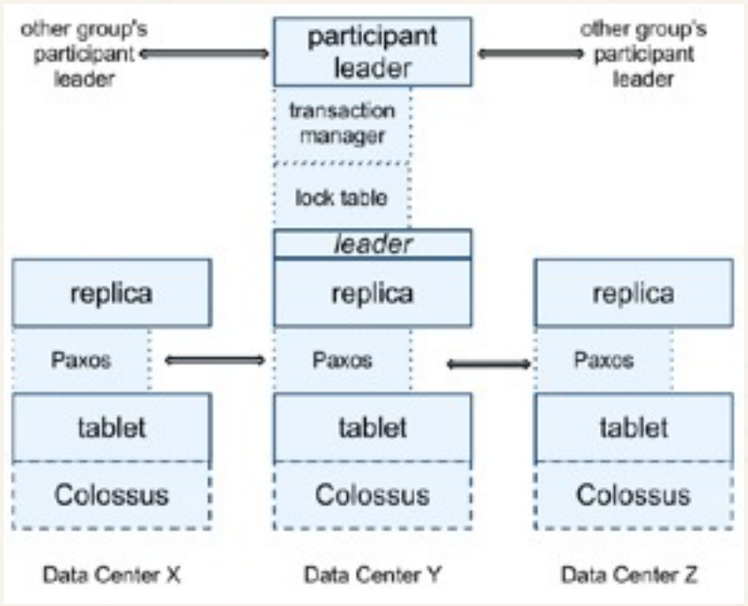


- O *universe master (u.m.)* e o *placement drive (p.d.)* são normalmente
- O *universe master* é um console que mostra informações sobre todas as zonas (para um *debugging* interativo);
- O *placement drive* manipula automaticamente o movimento dos dados através das zonas (na escala de tempo de minutos);
- O *p.d.* se comunica periodicamente com os *spanservers* para encontrar dados que precisam ser movidos, replicações atualizadas, balanceamento de carga;

Spanserver Software Stack

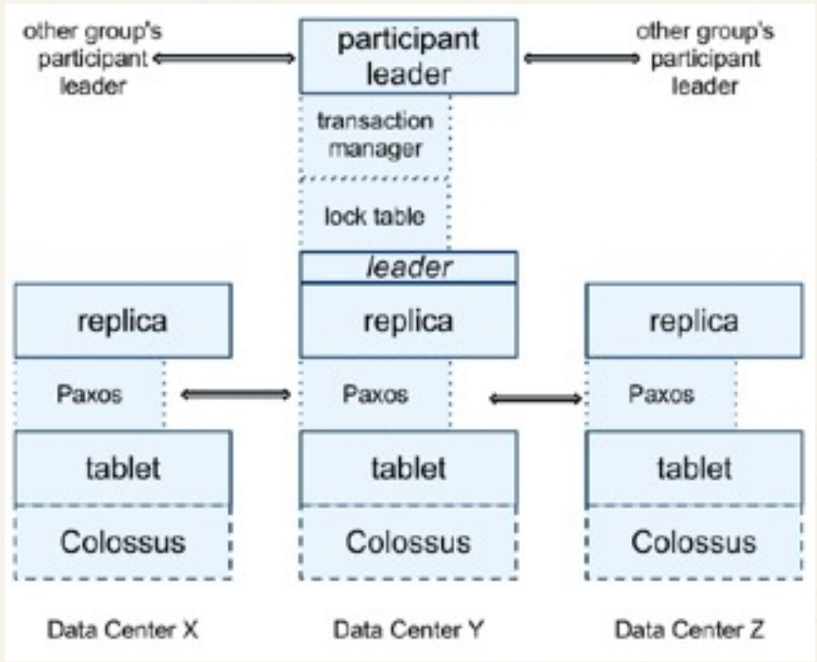


Spanserver Software Stack



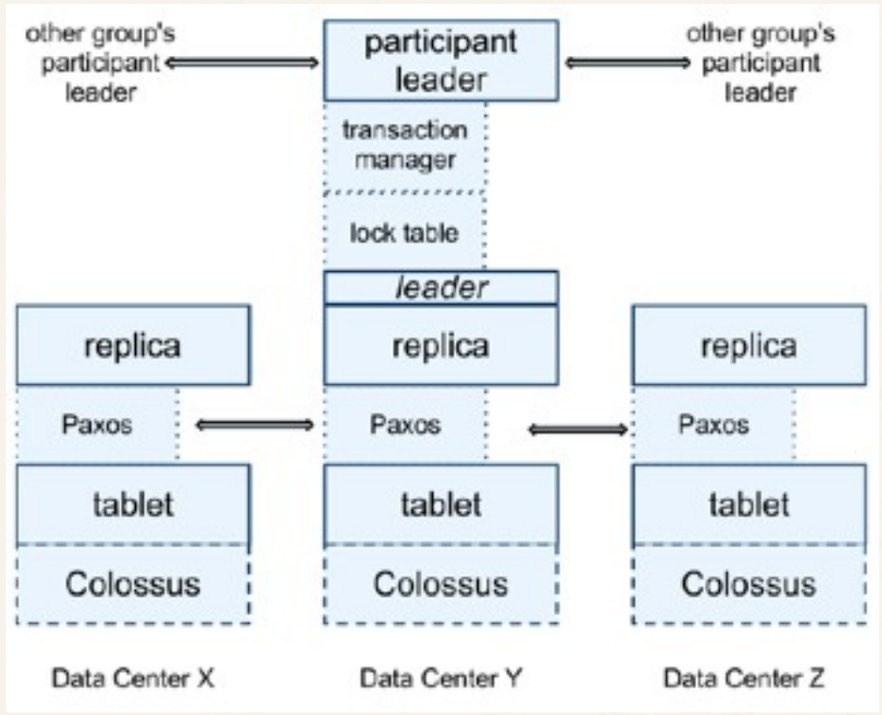
- Na base, cada *spanserver* é responsável por entre 100 e 1000 instâncias de uma estrutura de dados chamada de *tablet*;
- Um *tablet* é similar à abstração de tablet da Bigtable;
- Um estado da *tablet* é armazenado em um conjunto de arquivos como *B-tree* e um *log* de escrita - tudo em um sistema de arquivo distribuído chamado Colossus;

Spanserver Software Stack



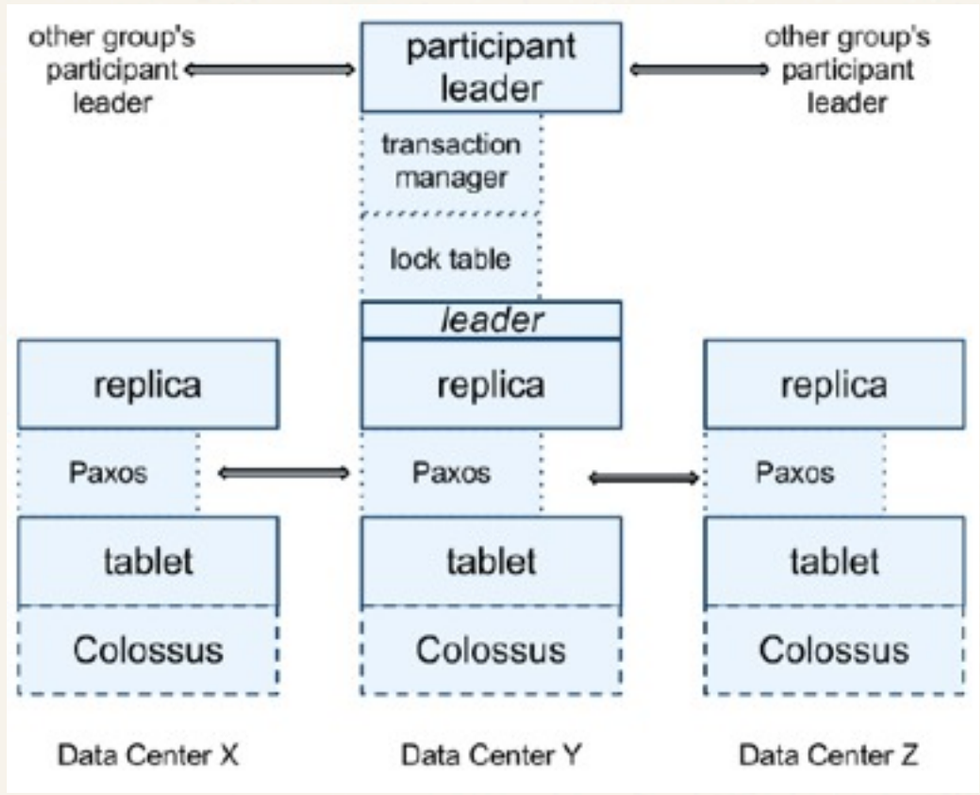
- Para suportar replicação, cada *spanserver* implementa uma simples máquina de estado Paxos no topo de cada *tablet*;
- Cada máquina de estado armazena seu metadado e *log* nos seus *tablet* correspondente;
- A implementação dos Paxos está *pipelined* de modo a melhorar o *throughput* do Spanner na presença de latências de WAN;

Spanserver Software Stack



- As máquinas de estado Paxos são usadas para implementar um pacote de mapeamento replicado consistentemente;
- O estado de mapeamento chave-valor de cada réplica é armazenado no seu *tablet* correspondente;
- O conjunto de réplicas é coletivamente um *Paxos group*;

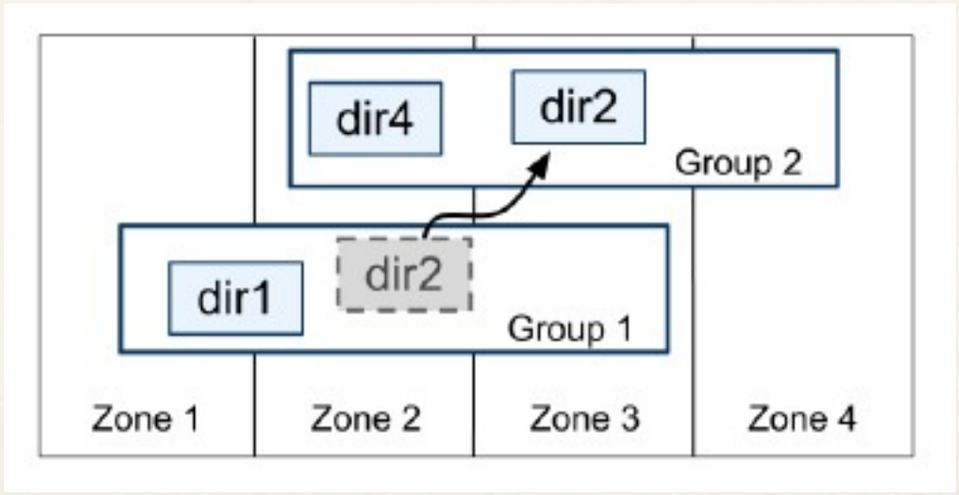
Spanserver Software Stack



- Em toda réplica que é líder, cada *spanserver* implementa uma *lock table* para implementar controle de concorrência;
- Em toda réplica que é líder, cada *spanserver* implementa também um *gerenciamento de transações* para suportar transações distribuídas.

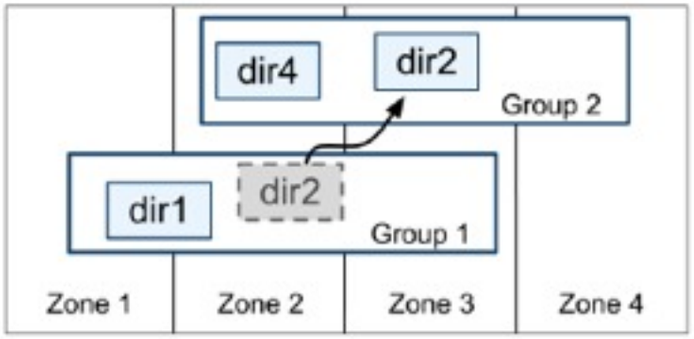
Diretórios e Localização

- No topo do pacote de mapeamento chave-valor, a implementação do Spanner suporta uma abstração chamada de diretório;
- Um diretório é um conjunto de chaves contínuas que compartilham um prefixo comum;
- Um diretório é a unidade de localização do dado. Todos os dados no diretório têm a mesma configuração de replicação;



- Diretórios são as unidades de movimentação dos dados entre os *Paxos groups*;

Diretórios e Localização



- Diretórios podem ser movidos enquanto operações de cliente são contínuas;
- Há uma estimativa de que um diretório de 50MB pode ser movido em poucos segundos;
- O fato de que um *Paxos group* pode conter múltiplos diretórios implica que um *tablet* Spanner é diferente de um *tablet* Bigtable;
- Um *tablet* Spanner é um container que pode encapsular múltiplas partições do *row space*;

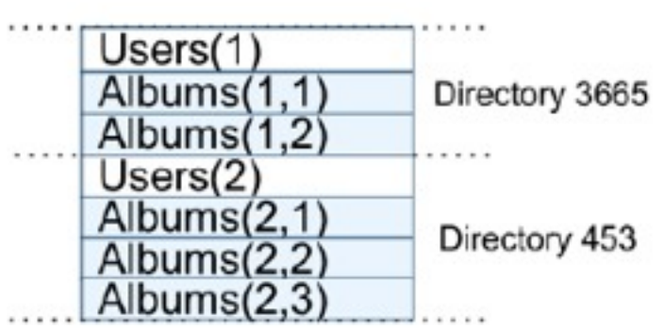
Modelo de Dados

- O Spanner fornece:
  - ✓ modelo de dados baseado em tabelas schmatizadas semi-relacionais;
  - ✓ linguagem de consulta;
  - ✓ transações de propósito geral;
- Uma aplicação cria um ou mais *databases* em um universo. Cada *database* pode conter um número ilimitado de tabelas *schematized*;
- As tabelas se parecem com com as tabelas dos *databases* relacionais, com linhas, colunas e valores;
- O modelo de dados do Spanner não é puramente relacional - as linhas devem ter nomes. Para toda tabela é necessário ter um conjunto ordenado de uma ou mais colunas *primary-key*;

Modelo de Dados


```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

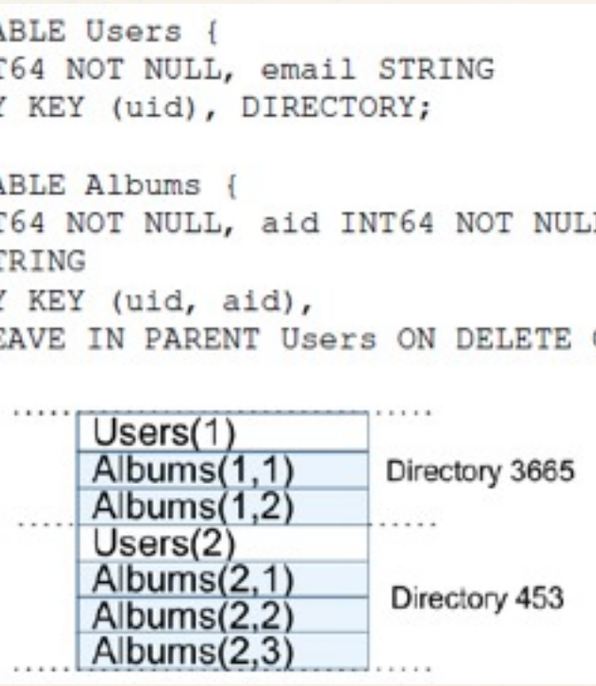


- Exemplo de um schema Spanner para armazenar fotos baseados em usários e albuns;
- Similar ao Megastore, porém todo *database* no Spanner deve ser particionado pelos clientes dentro de uma ou mais tabelas hierárquicas;

Modelo de Dados


```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```



- Aplicações clientes declaram as hierarquias nos *database schemas* via a declaração INTERLEAVE IN;

- A tabela no topo da hierarquia é uma tabela de diretório;

- ON DELETE CASCADE diz que deletando uma linha na tabela de diretório, deleta qualquer linha filha associada.

TrueTime API

Method	Returns
$TT.now()$	*TTinterval*: [*earliest*, *latest*]
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

- A tabela lista os métodos da API. TrueTime representa tempo como um *TTinterval*, o qual é um intervalo com tempo limitado de suspensão;
- Os *endpoints* de um *TTinterval* são do tipo *TTstamp*;
- O método *TT.now()* retorna um *TTinterval*;
- As referências de tempo subjacentes utilizados pelo TrueTime são GPS e *atomic clocks*.

Avaliação

- Medição de desempenho do Spanner em relação à replicação, às transações e à disponibilidade;
- Dados relacionados ao comportamento da TrueTime API;
- Estudo de caso em cima do primeiro cliente: F1;

- Microbenchmarks:

replicas	latency (ms)			throughput (Kops/sec)		
	write	read-only transaction	snapshot read	write	read-only transaction	snapshot read
1D	9.4±.6	—	—	4.0±.3	—	—
1	14.4±1.0	1.4±.1	1.3±.1	4.1±.05	10.9±.4	13.5±.1
3	13.9±.6	1.3±.1	1.2±.1	2.2±.5	13.8±3.2	38.5±.3
5	14.4±.4	1.4±.05	1.3±.04	2.8±.3	25.3±5.2	50.0±1.1

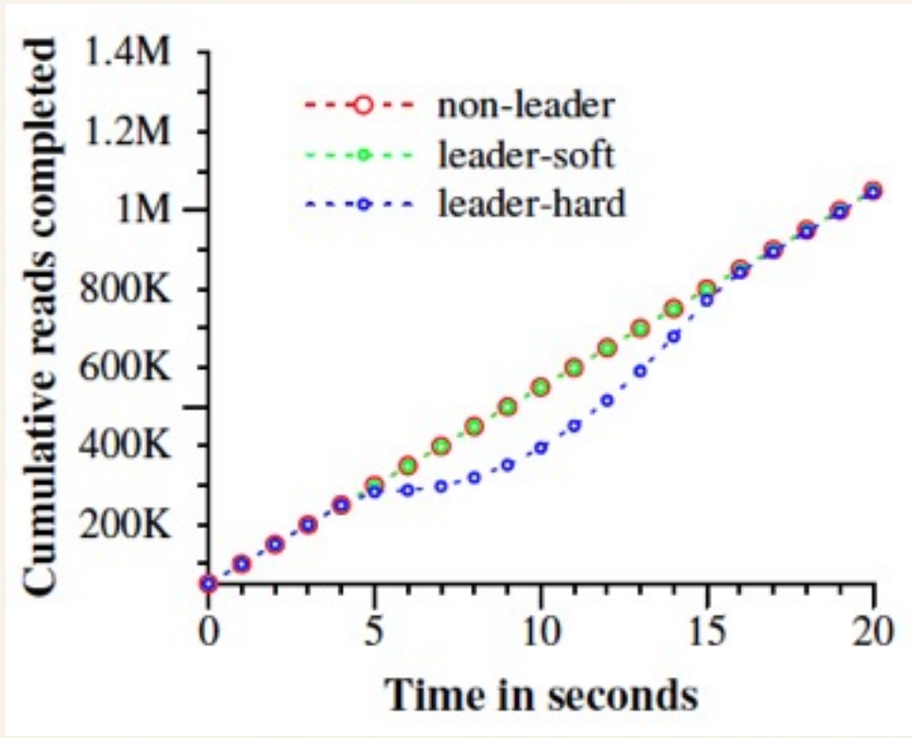
Avaliação

participants	latency (ms)	
	mean	99th percentile
1	17.0 ±1.4	75.0 ±34.9
2	24.5 ±2.5	87.6 ±35.9
5	31.5 ±6.2	104.5 ±52.2
10	30.0 ±3.7	95.6 ±25.4
25	35.5 ±5.6	100.4 ±42.7
50	42.7 ±4.1	93.7 ±22.9
100	71.4 ±7.6	131.2 ±17.6
200	150.5 ±11.0	320.3 ±35.1

- A tabela acima demonstra que commit two-phase pode ser dimensionado para um número razoável de participantes: se resume a um conjunto de experimentos executados em 3 zonas, cada uma com 25 spanservers;
- Até 50 participantes parece razoável (latências começam a aumentar visivelmente em 100 participantes);

Avaliação

- Disponibilidade:



- Benefícios de disponibilidade de execução do Spanner em múltiplos *datacenters*;
- Resultado de 3 experimentos na vazão na presença de falha no *datacenter*, todos sobrepostos na mesma escala de tempo;

Avaliação

- Estudo de caso F1:

# fragments	# directories
1	>100M
2–4	341
5–9	5336
10–14	232
15–99	34
100–500	7

- Ilustra a distribuição do número de fragmentos por diretórios no F1;
- Cada diretório corresponde tipicamente a um cliente na aplicação que roda sobre o F1.

Trabalhos Relacionados

- Megastore: Provinding Scalable, Highly Available Storage for Interactive Services [Armbrust et al., 2011];
- Amazon DynamoDB [2012];
- MapReduce: a flexible data processing tool [Dean and Ghemawat, 2010];
- HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads [Abouzeid, 2009];
- Consensus on transaction commit [Gray and Lamport, 2006];
- etc.

Trabalhos Futuros

- Migração do F1 Google's advertising do MySQL para o Spanner;
- Melhorias nas ferramentas de suporte, bem como nas ferramentas de *tuning* e desempenho;
- Melhorias nas funcionalidades e desempenho do sistema de *backup/restore*;
- Implementação da linguagem de *schema* do Spanner - manutenção automática de índices secundários;
- Permitir mudanças diretas das configurações dos Paxos;
- Mover de maneira coordenada e automatica os processos da aplicação-cliente entre *datacenters*.

Conclusões

- O Spanner combina e estende ideias a patir de duas comunidades de pesquisa: BD e linguagem de consulta;
- O Spanner faz mais do que simplesmente resolver o problema da replicação global: levar características de BD que faltam na Bigtable;
- Uso da TrueTime API apoiando aplicação distribuída.

FIM