ZooKeeper: Wait-free coordination for Internet-scale systems

Felipe Archanjo da Cunha Mendes
Felipe Sousa Pinheiro
Universidade Tecnológica Federal do Paraná
Bacharelado em Ciência da Computação

ZooKeeper: Wait-free coordination for Internet-scale systems

Patrick Hunt and Mahadev Konar Yahoo! Grid
{phunt,mahadev}@yahoo-inc.com
Flavio P. Junqueira and Benjamin Reed Yahoo! Research
{fpj,breed}@yahoo-inc.com

1. Introdução

APACHE
ZooKeeper™

1. Introdução

- Visão geral do serviço;
- API do Cliente;
- Garantias do Zookeeper;
- Exemplos de primitivas;
- Aplicações;
- Implementação;
- Avaliação;
- Trabalhos relacionados;
- Conclusão.

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

1. Introdução

- Aplicações distribuídas em larga escala requerem coordenação. A configuração, participação em grupos, eleição de líderes e bloqueios são formas comuns de coordenação;

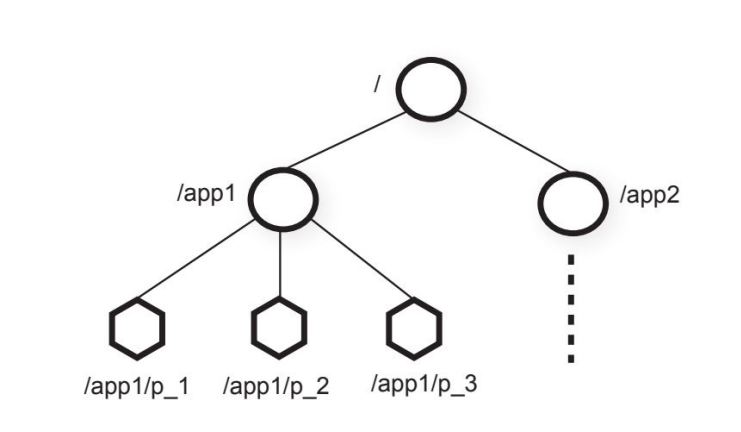
- Abordagem para a coordenação: Desenvolver serviços para diferentes necessidades de coordenação. O ZooKeeper é um serviço de coordenação que permite aos desenvolvedores implementar suas próprias primitivas;

- API do ZooKeeper: Expõe uma API que manipula objetos de dados hierarquicamente organizados como em sistemas de arquivos. A API não inclui primitivas de bloqueio;

- Garantias de ordem e consistência: O ZooKeeper garante a ordem FIFO das operações do cliente e implementa um protocolo de transmissão atômica baseado em líder para garantir a linearização das operações de atualização.

- Desempenho e cache: O ZooKeeper usa um mecanismo de observação para permitir que os clientes armazenem em cache dados sem gerenciar o cache diretamente. Os relógios ZooKeeper evitam problemas causados por clientes lentos ou defeituosos;

2. Visão geral do serviço

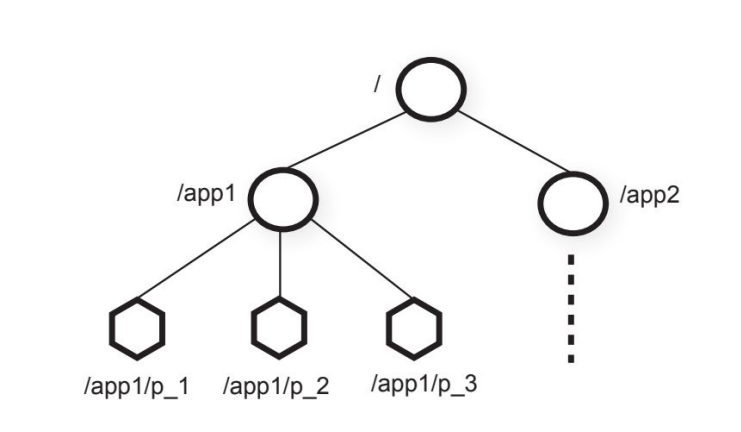


- O ZooKeeper fornece uma abstração de um conjunto de nós de dados (znodes) organizados hierarquicamente em um namespace.
- Os clientes manipulam esses znodes por meio da **ZooKeeperAPI**.
- Existem znodes regulares e efêmeros, e é possível definir um sinalizador sequencial ao criar um novo znode.

- **Modelo de dados:** O modelo de dados do ZooKeeper é semelhante a um sistema de arquivos simplificado, permitindo a alocação de subárvores para diferentes aplicativos e definição de direitos de acesso. Os znodes são mapeados para abstrações de metadados usados para coordenação.

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

2. Visão geral do serviço



- O ZooKeeper fornece uma abstração de um conjunto de nós de dados (znodes) organizados hierarquicamente em um namespace.
- Os clientes manipulam esses znodes por meio da **ZooKeeperAPI**.
- Existem znodes regulares e efêmeros, e é possível definir um sinalizador sequencial ao criar um novo znode.

- **Sessões:** Os clientes estabelecem uma sessão quando se conectam ao ZooKeeper e obtêm um identificador de sessão. As sessões têm um tempo limite e permitem que os clientes observem mudanças de estado e se movam transparentemente entre os servidores ZooKeeper.

3. API do cliente

- **create(path, data, flags):** Cria um novo znode com o nome especificado por "path", armazenando os dados contidos em "data[]". O parâmetro "flags" permite que o cliente selecione o tipo de znode (regular, efêmero) e defina a flag sequencial.

- **delete(path, version):** Exclui o znode especificado por "path" se sua versão corresponder à versão esperada especificada por "version".

- **exists(path, watch):** Verifica se o znode com o nome especificado por "path" existe. Retorna verdadeiro se o znode existir e falso caso contrário. A flag "watch" permite que o cliente defina um watch (observação) no znode.

- **getData(path, watch):** Retorna os dados e metadados associados ao znode especificado por "path", como informações de versão. Assim como no método "exists()", a flag "watch" permite que o cliente defina um watch no znode.

3. API do cliente

- **setData(path, data, version):** Grava os dados especificados por "data[]" no znode especificado por "path" se a versão do znode corresponder à versão esperada especificada por "version".

- **getChildren(path, watch):** Retorna o conjunto de nomes dos filhos de um znode especificado por "path".

- **sync(path):** Aguarda que todas as atualizações pendentes no início da operação sejam propagadas para o servidor ao qual o cliente está conectado. O parâmetro "path" é atualmente ignorado.

3. API do cliente

- **Métodos síncronos e assíncronos:** Todos os métodos têm versões síncronas e assíncronas disponíveis na API do ZooKeeper. A API síncrona é usada quando um aplicativo precisa executar uma única operação do ZooKeeper e não tem tarefas simultâneas. Já a API assíncrona permite que um aplicativo tenha várias operações pendentes do ZooKeeper e execute outras tarefas em paralelo. O cliente ZooKeeper garante que os retornos de chamada correspondentes a cada operação sejam invocados na ordem correta.

- **Uso de caminhos completos:** O ZooKeeper não utiliza identificadores para acessar znodes. Em vez disso, cada solicitação inclui o caminho completo do znode no qual a operação será realizada. Essa escolha simplifica a API e elimina a necessidade de o servidor manter estado adicional para atualizações subsequentes. Se a versão real do znode não corresponder à versão esperada, a atualização falhará com um erro de versão inesperada. Um número de versão de -1 indica que a verificação de versão não deve ser executada.

4. Garantias do ZooKeeper

- **Gravações linearizáveis:** Todas as solicitações que atualizam o estado do ZooKeeper são serializáveis e respeitam a precedência. Isso significa que as gravações são tratadas de forma linear, garantindo consistência e ordenação correta das operações.

- **Ordem do cliente FIFO:** Todas as solicitações de um determinado cliente são executadas na ordem em que foram enviadas pelo cliente. Isso garante que as operações de um cliente sejam processadas de forma sequencial e em ordem correta.

- **Definição de linearizabilidade:** O ZooKeeper adota uma definição modificada de linearizabilidade chamada linearizabilidade A (linearizabilidade assíncrona). Nessa definição, um cliente pode ter múltiplas operações pendentes, permitindo que várias operações sejam processadas em paralelo. O ZooKeeper garante a ordem correta das operações pendentes do mesmo cliente, seguindo a ordem FIFO.

- **Escalabilidade linear:** O ZooKeeper processa as solicitações de leitura localmente em cada réplica, permitindo que o serviço seja escalável linearmente à medida que mais servidores são adicionados ao sistema.

4. Garantias do ZooKeeper

- **Garantia de exclusividade de configuração:** O ZooKeeper pode ser usado para garantir a exclusividade de configuração em cenários onde um novo líder precisa alterar parâmetros de configuração em vários processos. O novo líder designa um znode especial como "ready", e os outros processos só usam a nova configuração quando esse znode existe. As alterações de configuração são realizadas de forma assíncrona e ordenada, garantindo que todos os processos vejam a configuração atualizada.

- **Garantia de ordenação de notificações:** O ZooKeeper garante que, se um cliente estiver observando uma alteração, o cliente verá o evento de notificação antes de ver o novo estado do sistema após a alteração. Isso evita que um cliente leia um estado inconsistente durante uma alteração em andamento.

- **Sincronização para leituras lentas:** O ZooKeeper fornece a solicitação de sincronização, que, quando seguida por uma leitura, constitui uma leitura lenta. A sincronização garante que um servidor aplique todas as gravações pendentes antes de processar a leitura, evitando a sobrecarga de uma gravação completa.

4. Garantias do ZooKeeper

- **Garantias de vivacidade e durabilidade:** Se a maioria dos servidores ZooKeeper estiver ativa e se comunicando, o serviço estará disponível. Além disso, se o serviço ZooKeeper responder com êxito a uma solicitação de alteração, essa alteração será persistente, mesmo em caso de falhas, desde que um quórum de servidores seja capaz de se recuperar.

5. Exemplos de primitivas

- Primitivas mais poderosas podem ser implementadas usando a API do ZooKeeper.

- O ZooKeeper é um serviço que não conhece essas primitivas, pois são implementadas no cliente usando a API do ZooKeeper.

- Algumas primitivas, como gerenciamento de configuração e associação de grupo, são isentas de espera, enquanto outras, como rendezvous, requerem que os clientes esperem por eventos.

5. Exemplos de primitivas

5.1 Gerenciamento de Configuração

- ZooKeeper pode implementar configuração dinâmica em aplicativos distribuídos.
- A Configuração é armazenada em um znode chamado zc.
- Os processos iniciam com o caminho completo de zc.
- Processos iniciais leem zc com sinalizador watch definido como verdadeiro.
- Se o zc for atualizado, os processos são notificados e leem a nova configuração.

5. Exemplos de primitivas

5.2 Rendezvous

- Usado em sistemas distribuídos quando a configuração final do sistema não é conhecida antecipadamente.
- Cliente cria um znode chamado zr para agendar processos mestre e de trabalho.
- Mestre preenche zr com informações sobre endereços e portas que está usando.
- Trabalhadores leem zr com watch definido como verdadeiro.
- Se zr ainda não foi preenchido, os trabalhadores esperam ser notificados quando zr for atualizado.

5. Exemplos de primitivas

5.3 Associação de Grupo

- Utiliza nós efêmeros para implementar a associação de grupo.
- Um znode chamado zg representa o grupo.
- Quando um membro do grupo é iniciado, ele cria um filho efêmero em zg.
- Informações do processo são colocadas nos dados do filho znode.
- Processos podem obter informações de grupo listando os filhos de zg.

5. Exemplos de primitivas

5.4 Bloqueios Simples

- O ZooKeeper pode ser usado para implementar bloqueios.
- Bloqueio é representado por um znode.
- Cliente tenta criar o znode com o sinalizador EFÊMERO para adquirir um bloqueio.
- Se a criação falhar, o cliente pode observar o znode para ser notificado se o líder atual morrer.
- Cliente libera o bloqueio excluindo o znode.

5. Exemplos de primitivas

5.5 Bloqueios Simples sem Efeito de Manada

- Alinha os clientes solicitando o bloqueio e cada cliente obtém o bloqueio por ordem de chegada.
- Uso do sinalizador SEQUENTIAL na criação ordena as tentativas de bloqueio.
- Cliente mantém o bloqueio se o znode do cliente tiver o menor número de sequência.
- Evita o efeito manada ativando apenas um processo quando um bloqueio é liberado.

5. Exemplos de primitivas

5.6 Bloqueios de Leitura/Gravação

- Procedimentos de bloqueio de leitura e gravação separados.
- Bloqueios de gravação usam znodes chamados write-.
- Bloqueios de leitura podem ser compartilhados.
- Apenas znodes de bloqueio de gravação anteriores impedem bloqueios de leitura.

6. Aplicações

6.1 The Fetching Service

Crawling de dados é essencial para motores de busca.
O fetching service faz parte do crawler do Yahoo. Consiste em um processo mestre que coordena outros processos de busca. Além disso, ele realiza uma checagem de saúde dos serviços para eleger um novo mestre caso necessário.

6. Aplicações

6.2 Katta

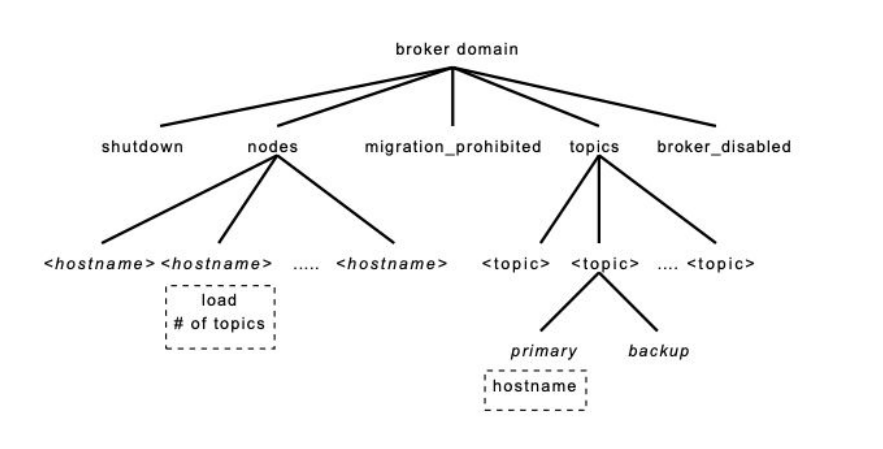
Serviço de indexação que utiliza shards. Um servidor mestre designa shards para servidores escravos. O mestre também pode falhar então também há eleições.
O Katta também utiliza o gerenciamento de configurações do zookeeper para atribuir shards para os servidores escravos.

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

6. Aplicações

6.3 Yahoo! Message Broker

É um sistema distribuído de publish subscribe no qual gerencia milhares de tópicos. Cada um dos tópicos é replicado usando um sistema de backup primário para garantir confiabilidade no envio das mensagens.

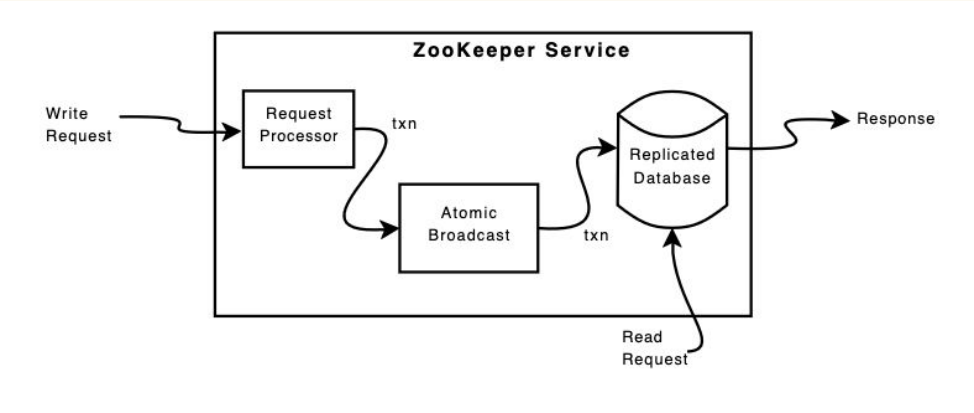


7. Implementação

O zookeeper possui uma alta disponibilidade através de réplica de dados em cada servidor que compõe o serviço:

Request Processor

Como a camada de mensagens é atômica, deve-se garantir que as réplicas locais nunca se divergem.
O Líder calcula o estado futuro para garantir a integridade.

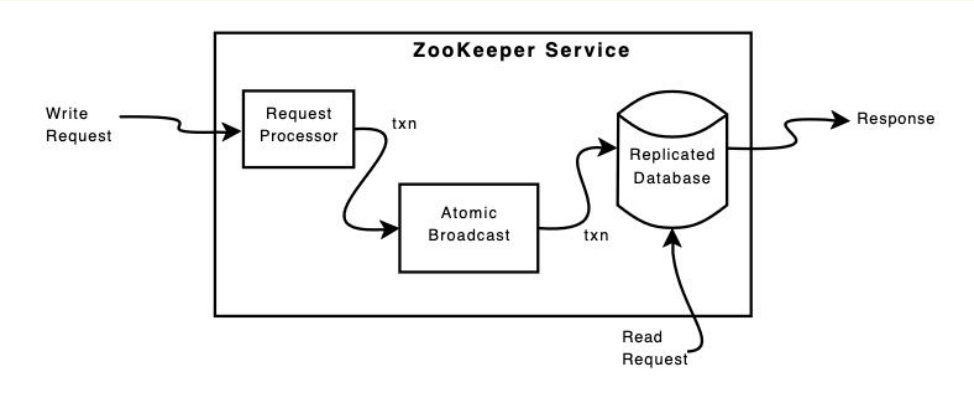


7. Implementação

O zookeeper possui uma alta disponibilidade através de réplica de dados em cada servidor que compõe o serviço:

Atomic Broadcast

O líder executa a solicitação e transmite a mudança para o estado do ZooKeeper através do Zab [24], um protocolo de transmissão atômica.



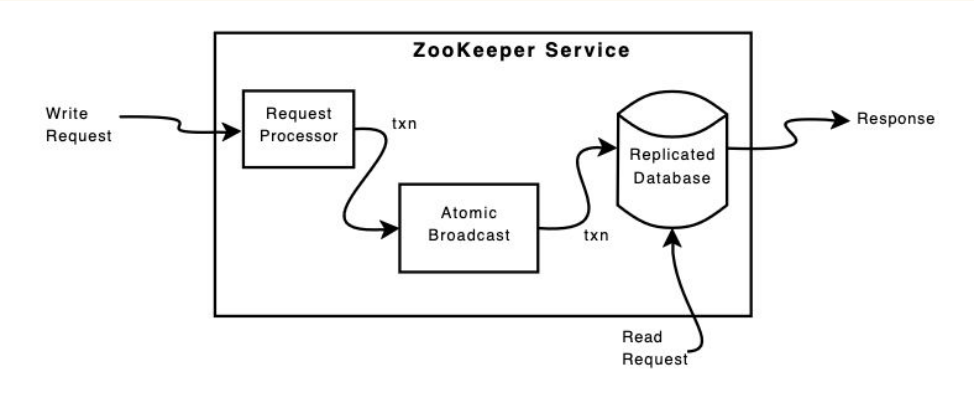
7. Implementação

O zookeeper possui uma alta disponibilidade através de réplica de dados em cada servidor que compõe o serviço:

Replicated Database

Cada réplica possui uma cópia em memória do estado do ZooKeeper. Quando um servidor do ZooKeeper se recupera de uma falha, ele precisa recuperar esse estado interno.

Recuperação feita através de snapshots.

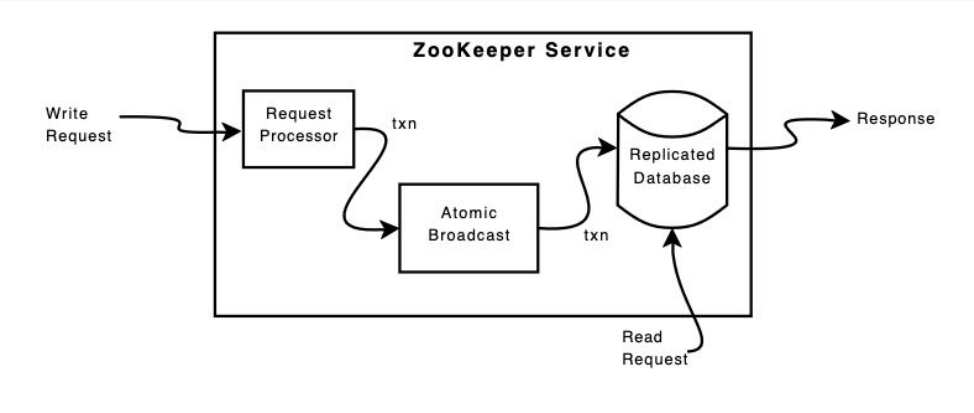


7. Implementação

O zookeeper possui uma alta disponibilidade através de réplica de dados em cada servidor que compõe o serviço:

Client-server Interactions

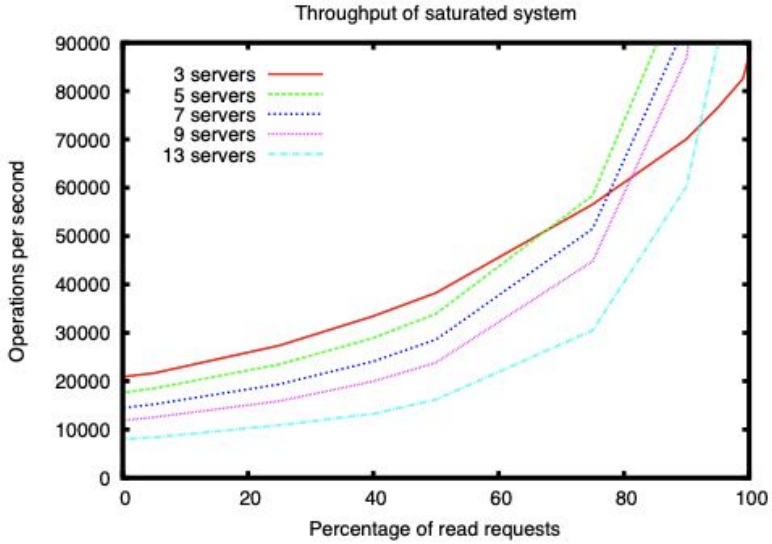
Quando um servidor processa uma solicitação de escrita, ele também envia e limpa notificações relacionadas a qualquer observação (watch) correspondente àquela atualização. Os servidores processam as escritas em ordem e não processam outras escritas ou leituras simultaneamente, garantindo uma sucessão rigorosa de notificações.



8. Avaliação

8.1 Taxa de Transferência

Request throttling para que os servidores não sejam sobrecarregados



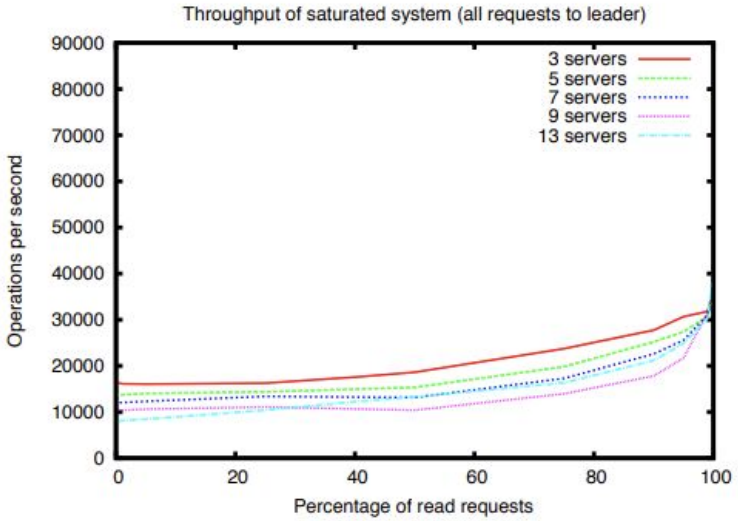
Testes:
50 Servidores com:
Xeon dual-core 2.1GHz
4GB de RAM

Servers	100% Reads	0% Reads
13	460k	8k
9	296k	12k
7	257k	14k
5	165k	18k
3	87k	21k

8. Avaliação

8.1 Taxa de Transferência

Desempenho em troca de confiabilidade, a taxa de transferência é aumentada particionando os dados do Zookeeper em múltiplos conjuntos



Testes:
50 Servidores com:
Xeon dual-core 2.1GHz
4GB de RAM

8. Avaliação

8.2 Latência de requisições

Teste de latência utilizando requisições de create

	Number of servers			
Workers	3	5	7	9
1	776	748	758	711
10	2074	1832	1572	1540
20	2740	2336	1934	1890

Table 2: Create requests processed per second.

9. Trabalhos relacionados

- Chubby (transações);
- ISIS (tolerância a falhas);
- Algoritmo Paxos;
- BoxWood;
- Sinfonia;
- Depspace.

9. Conclusões

- Resolução de problemas de coordenação internos do yahoo e externos;
- Apenas algumas aplicações foram citadas, porém existem diversas aplicações em larga escala que utilizam o zookeeper;
- Alcança uma taxa de transferência de centenas de milhares de requisições por segundo.