BeeFS: Um Sistema de Arquivos Distribuído POSIX Barato e Eficiente para Redes Locais

Carla A. Souza, Ana Clara Lacerda, Jonhnny W. Silva
Thiago Emmanuel Pereira, Alexandro Soares, Francisco Brasileiro[1]

[1]Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
58.429-900, Campina Grande, PB
Brasil

{carla,anaclara,jonhnny,thiagoepdc,alexandro}@lsd.ufcg.edu.br

fubica@dsc.ufcg.edu.br

Abstract. *BeeFS is a distributed file system that harnesses the free disk space of machines already deployed in the LAN. Like some special-purpose rack-aware file systems, it uses a hybrid architecture that follows a client-server approach for serving metadata and manage file replicas, and a peer-to-peer one for serving data. This characteristic allows BeeFS to aggregate the spare space of desktop disks to build a single logical volume on top of which a general purpose fully POSIX-compliant file system is implemented. BeeFS is not only more efficient than the state-of-practice that uses a dedicated server approach (eg. NFS), but also cheaper and naturally scalable.*

Resumo. *BeeFS é um sistema de arquivos distribuído que agrega o espaço ocioso dos discos de estações de trabalho de uma rede local. Utiliza uma arquitetura híbrida que mistura aspectos cliente-servidor para servir os metadados dos arquivos e as características da arquitetura entre-pares para prover os arquivos. Esse conjunto de características permite ao BeeFS agregar o espaço dos discos e construir um volume lógico de dados que é compatível com a norma POSIX. BeeFS não é somente mais eficiente do que sistemas de arquivos que utilizam servidores dedicados (ex. NFS), mas é, também, mais escalável e barato.*

1. Introdução

O BeeFS [Pereira et al. 2009] é um sistema de arquivos distribuído desenvolvido para atender a requisitos de escalabilidade e manutenabilidade pouco oferecidos por sistemas de arquivos distribuídos amplamente utilizados na prática, como NFS [Pawlowski et al. 1994] e Coda [Satyanarayanan et al. 1990]. O BeeFS é composto por um único servidor de metadados, que também atua como gerente de replicação; por diversos servidores de armazenamento de dados, que residem em estações de trabalho de uma LAN, promovendo a utilização de recursos de armazenamento ociosos e ampliando a capacidade de armazenamento de dados; e por diversos clientes, que implementam uma interface POSIX, provendo acesso transparente aos arquivos armazenados pelo sistema.

Os servidores de armazenamento, por residirem em estações de trabalho, estão sujeito a condições menos controladas de funcionamento. Para aumentar a confiabilidade

no serviço de acesso aos dados, o BeeFS implementa um modelo de replicação passiva não-bloqueante que mantém cópias dos dados em servidores de armazenamento distintos, assegurando um nível desejado de redundância indicado pelo administrador do sistema.

Associado ao mecanismo de replicação, o BeeFS explora a natureza entre-pares de seus componentes e procura alocar réplicas primárias em servidores que co-existam na máquina do cliente atual da réplica [Pereira et al. 2009]. Esta estratégia de alocação, ao favorecer o acesso local ao dados em detrimento do acesso remoto, minimiza o tempo de resposta das operações bem como a banda passante na LAN.

O BeeFS foi implementado no nível de usuário utilizando a linguagem de programação Java. A junção entre aplicativo no nível de usuário e os módulos de sistemas de arquivos do GNU/Linux foi feito via FUSE [FUSE 2008], o qual foi escolhido pela facilidade de implementação.

Nesse trabalho, será demonstrado o BeeFS em produção e uma descrição de um exemplo de uso que se beneficia da intrínseca distribuição dos dados para realizar processamento paralelo. Inicialmente, na Seção 2, falamos sobre a arquitetura, detalhando os componentes, tolerância a falhas, atributos dos arquivos e a segurança de armazenamento dos dados. Na Seção 3 mostramos uma avaliação de desempenho e escalabilidade feita comparando-o ao NFS por meio da execução de *benchmarks*. Uma demonstração de uso do BeeFS é descrita na Seção 4. Por fim, considerações finais acerca do trabalho são apresentadas na Seção 5.

2. Arquitetura

O BeeFS é modularizado em três componentes principais: um único servidor de metadados (*queenbee*), servidores de dados (*honeycombs*) e clientes (*honeybees*). Estes componentes seguem um modelo de arquitetura híbrido que mistura aspectos dos sistemas cliente-servidor e entre-pares, o que permite i) reduzir a carga de trabalho no componente centralizado e ii) atender o aumento da demanda de forma mais granular, através da adição de novos servidores de dados.

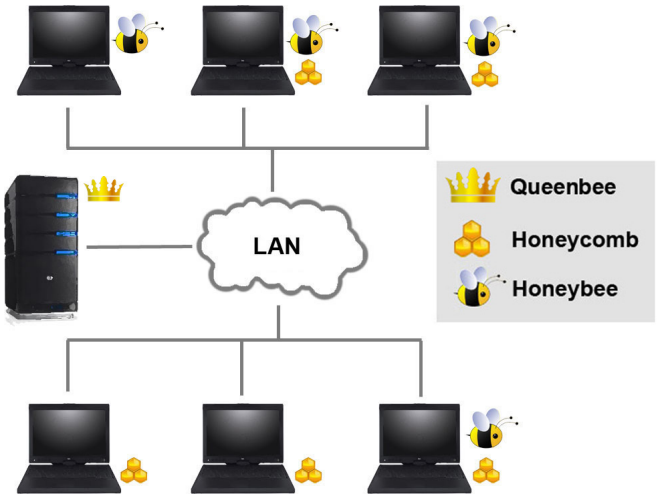

Figura 1. Visão Geral dos Componentes do BeeFS

O servidor *queenbee* é instalado, preferencialmente, num computador dedicado.

Ele é responsável pelo serviço de nomes, controle de acesso, coordenação do mecanismo de replicação e escalanomanento de réplicas. Por outro lado, não se envolve com o armazenamento dos dados brutos. Clientes *honeybee* o contactam para obter a localização dos servidores *honeycomb* os quais armazenam os arquivos. Depois disso, as transferência de dados são feitas diretamente entre *honeybee* e *honeycombs*.

Servidores *honeycomb* colaborativamente armazenam os dados servidos pelo BeeFS. É importante ressaltar que este componente provê apenas primitivas básicas de acesso a dados brutos, o mapeamento lógico entre uma coleção de dados brutos e um grupo de replicação, além do mapeamento entre um grupo de replicação e a visão lógica de arquivos e diretórios é mantida pelo servidor *queenbee*. Além de prover o acesso aos dados aos clientes, os *honeycombs* podem trocar dados entre si, sob a coordenação do servidor *queenbee*, durante o processo de atualização das réplicas.

Já o componente *honeybee* permite que os processos dos usuários tenham acesso aos arquivos. A localização dos arquivos não é mantida pelos *honeybees*, visto que esta informação não é estática. Para obter esta informação, os clientes usam o serviço de nomes, provido pelo servidor *queenbee*. Após conseguir a localização dos arquivos, os clientes contactam os servidores de dados (*honeycombs*) para realizar operações de escrita/leitura.

2.1. Tolerância a falhas

Assim como outros sistemas de arquivos, o principal mecanismo para tolerância a falhas no BeeFS é a realização de *backups* periódicos. Entretanto, dada à natural redundância de sua arquitetura, o BeeFS implementa alguns mecanismos adicionais de tolerância a falhas que podem evitar que dados sejam perdidos mesmo depois que falhas permanentes afetem tanto o *queenbee* quanto os *honeycombs*.

Para tolerância a falhas no componente *honeycomb*, o BeeFS implementa uma estratégia de replicação passiva não-bloqueante [Soares et al. 2009]. Segundo esse modelo ilustrado na Figura 2, o componente *honeybee* sempre faz operações dos arquivos no servidor *honeycomb* primário e posteriormente as atualizações dos arquivos são propagadas para os secundários. Os arquivos armazenados no BeeFS possuem, além dos artributos definidos na norma POSIX, atributos extendidos que armazenam informações utilizadas pelo BeeFS. Dentre essas informações, são armazenadas: uma *flag* indicando se o arquivo é a cópia primária, o número de versão do arquivo e o seu nível de replicação.

O servidor *queenbee* é responsável pela coordenação das atualizações das réplicas secundárias. Isso mantém consistente a visão do estado dos grupos de réplicas associados a cada arquivo do sistema. Essa visão possui a versão de cada réplica armazenada nos servidores *honeycomb*. Segundo a semântica POSIX, o cliente *honeybee* envia uma chamada de fechamento de arquivo para o servidor *queenbee* o qual atualiza sua visão sobre as réplicas. No final da chamada, o servidor *queenbee* escalona o processo de propagação do conteúdo da réplica primária para as demais. O tempo dessa propagação é definido pelo administrador do sistema por meio do parâmetro “tempo de coerência”.

O componente *queenbee* é, também, responsável pelo monitoramento dos servidores *honeycomb*. Quando é detectada a falha de um *honeycomb*, o grupo de replicação deve ser reorganizado. Um novo servidor deve substituir o *honeycomb* comprometido no grupo de replicação.

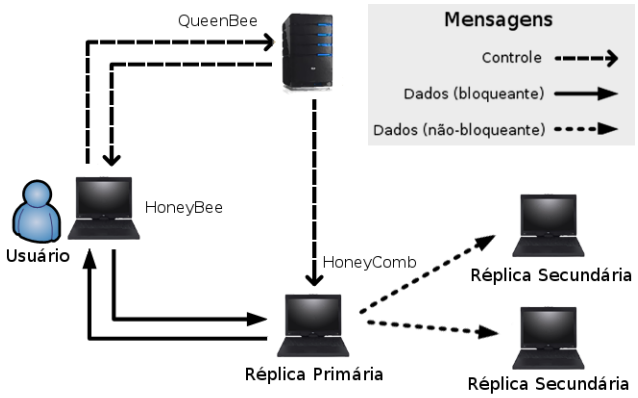

Figura 2. Modelo de Replicação BeeFS

Com relação ao servidor *queenbee*, existe a possibilidade de ocorrer falhas que comprometam permanentemente o servidor. Um tipo de falha do servidor *queenbee* tornaria indisponível permanentemente todas as referências para os arquivos armazenados no sistema, tornando-os inacessíveis. Logo, o BeeFS possui um modelo de armazenamento de metadados tolerante a falhas. De acordo com esse modelo, réplicas dos metadados armazenados no servidor *queenbee* também são mantidas pelos servidores *honeycombs*.

Quando ocorre uma falha no servidor *queenbee*, BeeFS assume um modo de recuperação de falha do sistema. Neste modo de recuperação, a checagem da consistência é realizada por meio de checagens nas cópias redundantes dos metadados localizados nos *honeycombs*. Desta forma, é possível restaurar o estado no qual o sistema se encontrava antes da falha ocorrer para a maioria dos casos [Soares et al. 2009].

2.2. Segurança

BeeFS dispersa arquivos entre os *desktops* da LAN. Esses arquivos devem ser protegidos contra ataques internos. Ambos privacidade e integridade podem ser comprometidos por usuários maliciosos. Farsite [Adya et al. 2002], um sistema de arquivo que também utiliza o espaço ocioso do disco das máquinas de redes corporativas, resolve esse problema utilizando criptografia.

Para garantir privacidade e integridade dos arquivos armazenados, assim como Farsite, nosso sistema incorpora criptografia dos dados armazenados em conjunto com o mecanismo de controle de acesso provido pelo sistema de arquivos local que é utilizado pelo servidor *honeycomb*. No caso do Sistema de Arquivos Linux, os dados brutos são mantidos com permissão de escrita e leitura apenas para o usuário do sistema que executa o BeeFS. Assim, privacidade e integridade podem ser facilmente garantidas assumindo que a permissão para cada arquivo armazenado no servidor *honeycomb* é definida de tal forma que apenas os componentes do BeeFS podem acessá-los.

3. Avaliação de desempenho

Nessa seção foi feita a análise de performance do BeeFS, comparando-o contra o NFS. O NFS representa o estado-da-prática para um serviço de sistema de arquivos numa LAN, e, portanto, é uma boa escolha para comparação. A seguir, é descrito o *benchmark* utilizado

para executar os testes de performance, apresentação dos resultados obtidos e análise dos mesmos.

O *benchmark* Andrew foi utilizado para comparação entre o NFS e BeeFS, o qual simula o *workload* de desenvolvimento de um *software*. Ele recebe como entrada a raiz de uma árvore localizada em um sistema de arquivos diferente do que está sendo testado. Ele é composto por 5 fases: `mkdir`, `copy`, `stat`, `grep` e `compile`.

1. A fase **Mkdir** cria a hierarquia de diretórios do código-fonte original no sistema de arquivos que está sendo testado;
2. A fase **Copy** copia todos os arquivos da árvore original para a árvore criada;
3. A fase **Stat** busca o *status* de todos os arquivos na árvore sem examinar seu conteúdo;
4. A fase **Grep** lê todo o conteúdo dos arquivos criados durante a fase de cópia;
5. A fase **Compile** compila e liga os arquivos.

O *benchmark* Andrew foi executado recebendo como entrada a hierarquia de arquivos que compõem o código fonte do aplicativo Nagios 3.0.6 (disponível em `http://www.nagios.org/`), o qual contém 21 diretórios, 515 arquivos e 9 Mbytes de informação.

Foram consideradas duas configurações para os experimentos com o BeeFS e uma configuração utilizando NFS, que são descritas a seguir:

- **BeeFS co-alocado**, a qual contém o servidor *queenbee* rodando numa máquina e o cliente *honeybee* e o servidor *honeycomb* co-alocados em outra máquina;
- **BeeFS não co-alocado**, a qual consiste no servidor *queenbee*, o cliente *honeybee* e o servidor *honeycomb* instalados em máquinas diferentes;
- **NFS**, a qual contém o servidor NFS executando numa máquina e o cliente NFS em outra máquina.

Todas as medidas de performance foram feitas em máquinas com processadores de dois cores 3.2 GHz Intel e 2 Gbytes de memória RAM cada, executando Ubuntu 9.04 *kernel* $2.6.28-11$-vserver. Todos os nós foram conectados através de uma rede 100 Mbit Ethernet que foi isolada do tráfego pesado durante a execução do experimento.[1]

A Tabela 1 mostra o resultado da execução do *benchmark* para as 3 configurações descritas, com 95% de nível de confiança testado estatisticamente.

A Tabela 1 também exibe o percentual de melhoria do tempo de execução para BeeFS Co-alocado e BeeFS Não Co-alocado em relação ao NFS. Para todas as configurações BeeFS, ambas as fases **Mkdir** e **Stat** possuem melhoria semelhantes — em torno de 40% para `mkdir` e 26% para `stat` —, porque essas fases consistem em apenas um fluxo de requisição assíncrona para o servidor *queenbee*.

Por outro lado, as fases de leitura e escrita intensa em disco (`copy` e `grep`) tiveram melhorias diferentes, refletindo o benefício da co-alocação; enquanto configurações co-alocadas realizam operações no disco local, máquinas não co-alocadas e NFS transmitem os dados pela rede.

[1]Alguns serviços distribuídos como o **NTP** e **IMAP** estavam rodando durante a execução do experimento. Entretanto, o tráfego de rede gerado por esses serviços não é substancial e não gera impacto significativo nos resultados do experimento.

Tabela 1. Média do tempo de execução (em ms) para cada fase individual para cada configuração: NFS, BeeFS co-alocado e BeeFS não co-alocado.

Fase	NFS	BeeFS	
		BeeFS co-alocado	BeeeFS não co-alocado
Mkdir	29.4	17.8 (40%)	18.4 (37%)
Copy	9,118.5	2,362.3 (74%)	3,954.8 (56%)
Stat	2,434.4	1,794.9 (26%)	1,712.9 (29%)
Grep	3,073.2	2,190.6 (29%)	2,481.5 (20%)
Compile	43,075.0	38,537.3 (10%)	43,472.5 (-1%)
Total	57,730.5	44,902.9 (22%)	51,640.1 (10%)

4. Demonstração

4.1. Instalação do Sistema de Arquivos BeeFS

A instalação do BeeFS é simples e pode ser feita sem a necessidade de permissão do administrador do sistema operacional pois sua execução pode ser feita no espaço do usuário. O BeeFS está disponível em `http://www.lsd.ufcg.edu.br/beefs` e a instalação é feita por meio do desempacotamento do software. Feito isso, o BeeFS ocupa 2.4 Mbytes quando instalado e sua estrutura está organizada da seguinte forma:

- **Diretório bin**, o qual possui os executáveis do BeeFS;
- **Diretório conf**, no qual estão localizados os arquivos de configuração para cada componente e arquivo de configuração do ambiente a ser instalado o BeeFS;
- **Diretório lib**, o qual possui a implementação do BeeFS e as bibliotecas das quais ele depende.

Após a instalação, deve ser feita a configuração dos componentes que deseja-se executar na máquina. Para a configuração do componente *honeybee* é necessário o preenchimento do endereço da máquina que possui o servidor *queenbee* instalado e qual será o diretório que disponibilizará o sistema de arquivos BeeFS. A configuração do componente *honeycomb* deve conter qual o diretório no qual serão armazenados os arquivos e deve indicar, como no *honeybee*, qual a máquina que possui o componente *queenbee*. Já a configuração do componente *queenbee* deve conter qual será o diretório que armazenará os metadados do sistema. Esses arquivos de configuração são, respectivamente, `honeybee.conf`, `honeycomb.conf` e `queenbee.conf`

A inicialização dos serviços é feita apenas executando o arquivo `beefs`, localizado no diretório `bin`, e indicando qual o serviço que se deseja iniciar: *queenbee*, *honeycomb* e *honeybee*.

Após esse procedimento, o BeeFS está pronto para uso e o acesso aos arquivos é dado a partir do ponto de montagem informado no arquivo de configuração do componente *honeybee*.

4.2. Processamento paralelo de forma simples com o BeeFS

Além do habitual uso de um sistema de arquivos, o BeeFS pode ser naturalmente utilizado por aplicações de processamento distribuído sem apresentar degradação de desempenho.

Isto é possível graças a uma estratégia de armazenamento distribuído que dispersa os arquivos entre os computadores da LAN.

Visando demonstrar a diferença de desempenho entre o BeeFS e o NFS (versão 3), foi elaborado um experimento que realiza um conjunto de tarefas independentes, as quais acessam simultaneamente o sistema de arquivos. Neste experimento será executado em 2 cenários construídos. O primeiro consiste de um servidor NFS instalado em uma máquina que compartilha arquivos com outras 4 máquinas, as quais possuem uma instalação do cliente NFS. Dentre as opções mais relevantes das configurações do cliente e servidor NFS, destacam-se: transferência de dados pela rede via conexões TCP, *buffers* de leitura e escrita com tamanho de 8192 *bytes*, comunicação assíncrona para escrita e *cache* de atributos habilitado. O segundo cenário consiste de um servidor *queenbee* em uma máquina e outras 4 máquinas com a instalação dos componentes *honeybee* e *honeycomb*.

Todas as máquinas utilizadas (remotamente) em ambos os cenários possuem a seguinte configuração: 2.67 GHz Intel com 16 cores e 8 Gbytes de memória RAM cada, executando Ubuntu 9.10 kernel 2.6.31−20−vserver, conectadas por uma rede Ethernet de 100 Mbit.

Um exemplo típico de uso de um sistema de arquivos é a análise do conteúdo de documentos de texto puro buscando a existência de um determinado conjunto de caracteres. Essa operação pode ser executada por meio do comando `grep` do GNU/Linux. No experimento, utilizamos o `grep` para buscar a ocorrência da palavra `SBRC` em um conjunto de arquivos texto com aproximadamente 500 Mbytes cada.

A busca no conteúdo desses arquivos será executada de forma distribuída com o comando `grep`. A distribuição foi realizada utilizando o bashreduce [Frey 2009] — uma implementação na linguagem *bash* do modelo map/reduce [Dean and Ghemawat 2008] que utiliza aplicativos shell do GNU/Linux. Essa operação foi encapsulada e denominada como `brgrep`. Para o exemplo citado anteriormente, a busca pode ser executada da seguinte forma:

```
$> brgrep SBRC inputfiles.txt
```

Listing 1. Exemplo de busca com `brgrep`.

Executando previamente esse caso de uso, no primeiro cenário a execução no sistema de arquivos NFS durou 43 minutos e 29 segundos, enquanto o segundo cenário, que utiliza o BeeFS durou 10 minutos e 12 segundos.

O desempenho diferenciado do BeeFS em relação ao NFS foi ocasionado pela co-alocação dos dados, que possibilitou o acesso direto aos arquivos do disco local da máquina. Por outro lado, o NFS necessitou do enlace de rede da LAN para acessar os arquivos armazenados no servidor NFS.

5. Conclusão

Neste trabalho, apresentamos o sistema de arquivos distribuído BeeFS, que utiliza o espaço ocioso do disco das máquinas instaladas em uma LAN para construir uma solução de armazenamento eficiente e escalável de forma barata.

A arquitetura híbrida que utiliza um servidor centralizado para armazenamento de metadados e gerenciamento de operações de réplicas dos arquivos, e a arquitetura entre-

pares para compartilhamento dos arquivos armazenados no sistema associado à estratégia de armazenamento co-alocado dos dados, permite a alta eficiência do sistema de arquivos. O uso dos discos das estações de trabalho e a fácil instalação do *software* permite que a capacidade de armazenamento cresca naturalmente quando novas máquinas que são adicionadas na LAN.

Como dito anteriormente, BeeFS implementa a API POSIX para sistemas de arquivos; isto é especialmente importante por motivo de compatibilidade com aplicações. A implementação foi validada por meio da suíte de testes de sistemas de arquivos Pawel Jakub Dawidek's POSIX mantida por Tuxera [POSIX 2009] e foi executada com sucesso em todos os 3.061 testes que compreendem a suíte, dando-nos confiança que o BeeFS é, realmente, POSIX compatível.

Atualmente, uma versão estável do BeeFS está em produção no Laboratório de Sistemas Distribuídos, disponibilizando um sistema de arquivos com mais de 1 Tbytes de espaço agregado para aproximadamente 30 pessoas. O serviço está em operação há mais de três meses. A documentação e o *software* para *download* estão disponíveis em `http://www.lsd.ufcg.edu.br/beefs`.

Referências

Adya, A., Bolosky, W., Castro, M., Chaiken, R., Cermak, G., Douceur, J., Howell, J., Lorch, J., Theimer, M., and Wattenhofer, R. (2002). Farsite: Federated, available, and reliable storage for an incompletely trusted environment.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Frey, E. (2009). Mapreduce bash script. `http://blog.last.fm/2009/04/06/mapreduce-bash-script`.

FUSE (2008). Filesystem in userspace. `http://fuse.sourceforge.net`.

Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D., and Hitz, D. (1994). NFS version 3 design and implementation. In *Proceedings of the Summer USENIX Conference*, pages 137–152.

Pereira, T. E., Silva, J. W., Soares, A. S., and Brasileiro, F. V. (2009). Beefs: A cheaper and naturally scalable distributed file system for corporate environments. Technical report, Federal University of Campina Grande - Distributed Systems Lab.

POSIX (2009). Posix test suite. `http://www.tuxera.com/community/posix-test-suite/`.

Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E., and Steere, D. (1990). Coda: a highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459.

Soares, A. S., Pereira, T. E., Silva, J. W., and Brasileiro, F. V. (2009). Um modelo de armazenamento de metadatados tolerante a falhas para o DDGfs. In *WSCAD-SSC 2009: Proceedings of the 10th Computational Systems Symposium*.