Armazenamento distribuído de dados e checkpointing de aplicações paralelas em grades oportunistas*

Autor: Raphael Y. de Camargo[1]
Orientador: Prof. Dr. Fabio Kon[1]

[1]Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo (USP), São Paulo-SP, Brasil
{rcamargo,kon}@ime.usp.br

Resumo. *Grades computacionais oportunistas utilizam recursos ociosos de máquinas compartilhadas para executar aplicações que necessitam de um alto poder computacional e/ou trabalham com grandes quantidades de dados. Neste trabalho, projetamos, implementamos e avaliamos uma infra-estrutura de software que permite a execução destas aplicações em grades oportunistas. Esta infra-estrutura é constituída por: (1) um mecanismo de tolerância a falhas baseado em* checkpointing *que permite a execução de aplicações paralelas mesmo com a presença de falhas em nós de execução e (2) um middleware, denominado OppStore, que permite a criação de uma infra-estrutura de armazenamento distribuído de dados de baixo custo e que utiliza o espaço livre em disco de máquinas compartilhadas da grade. Avaliamos nossa abordagem através de simulações e experimentos em redes de grande área.*

1. Introdução

Existe uma grande classe de aplicações que necessitam de um alto poder computacional e trabalham com grandes quantidades de dados. Estas aplicações incluem o seqüenciamento de genes, enovelamento de proteínas, análise de sinais (ex: SETI), análises financeiras, mineração de dados, física de partículas e simulações em engenharia. Mas cientistas, pesquisadores, analistas e engenheiros muitas vezes não têm acesso a uma infra-estrutura computacional que lhes permita a execução destas aplicações, normalmente por trabalharem em instituições que dispõem de recursos limitados. Por outro lado, estas mesmas instituições normalmente possuem centenas ou milhares de máquinas utilizadas pelos seus membros e que permanecem ociosas pela grande maioria do tempo. Se pudéssemos utilizar os ciclos computacionais ociosos e o espaço livre em disco destas máquinas, seria possível executarmos uma parte significativa destas aplicações.

Grades computacionais oportunistas [Thain et al. 2002, Goldchleger et al. 2004, de Camargo et al. 2006b] foram desenvolvidas com o objetivo de utilizar o tempo ocioso de máquinas compartilhadas para realizar computação útil, de modo a aumentar o poder computacional de uma instituição sem a necessidade de adquirir hardware adicional. Aplicações são executadas nas máquinas apenas quando estas estão ociosas, de modo a não alterar a Qualidade de Serviço percebida pelo dono da máquina. O InteGrade[1] [Goldchleger et al. 2004] é um middleware que permite a criação de grades

*Texto da tese disponível em http://www.ime.usp.br/~rcamargo.
[1]Disponível em http://www.integrade.org.br.

computacionais oportunistas. Ele é organizado como uma federação de aglomerados, onde cada aglomerado contém máquinas que disponibilizam seus recursos ociosos para utilização por aplicações da grade.

Mas garantir a execução robusta de aplicações paralelas em máquinas não-dedicadas pertencentes a um ambiente dinâmico e heterogêneo, como o de grades oportunistas, é uma tarefa difícil. Máquinas podem falhar, ficar indisponíveis ou mudar de ociosas para ocupadas inesperadamente, comprometendo a execução das aplicações. Para tal, *checkpoints* contendo o estado de uma aplicação podem ser periodicamente gerados e armazenados, permitindo a reinicialização da aplicação, em caso de falha em um de seus processos, a partir de um estado intermediário de sua execução. Além disso, no caso de aplicações paralelas, o mecanismo precisa considerar as dependências entre os processos da aplicação ao obter seu estado global.

Os *checkpoints* gerados precisam ser salvos em um meio de armazenamento estável. Além disso, aplicações da grade tipicamente manipulam grandes quantidades de dados e necessitam de uma infra-estrutura de armazenamento de dados confiável, de alta capacidade e acessível de qualquer ponto da grade. A solução imediata seria instalar servidores dedicados para o armazenamento dos dados de aplicações. Mas para tal teríamos que manter estes servidores dedicados, que além do custo de aquisição, geram calor, consomem energia, utilizam espaço e precisam ser gerenciados. Ao mesmo tempo, grades oportunistas são compostas por máquinas compartilhadas, que tipicamente possuem quantidades significativas de espaço livre em disco. Utilizar estes recursos ociosos permitiria que obtivéssemos grandes quantidades de espaço de armazenamento a um baixo custo e sem a aquisição de hardware extra.

1.1. Principais contribuições

Neste trabalho desenvolvemos uma infra-estrutura de software que permite a utilização de máquinas não-dedicadas tanto para a execução de aplicações paralelas de longa duração como para o armazenamento de dados. Esta infra-estrutura é importante para viabilizar a utilização prática de grades computacionais oportunistas.

As principais contribuições científicas obtidas foram: (1) usamos reflexão computacional para instrumentar aplicações paralelas BSP para gerar *checkpoints* portáveis, (2) analisamos diversas estratégias para o armazenamento de *checkpoints* de aplicações paralelas, (3) propusemos o conceito de identificadores virtuais, que permitem realizar o balanceamento dinâmico de carga entre nós heterogêneos utilizando como base a infra-estrutura de roteamento do Pastry, (4) projetamos e implementamos o middleware OppStore, que utiliza o espaço livre em disco das máquinas provedoras de recursos para o armazenamento distribuído de dados de aplicações da grade e (5) avaliação experimental e por simulação da viabilidade do uso do espaço livre em disco de máquinas ociosas para armazenar dados de aplicações.

2. Execução Tolerante a Falhas de Aplicações Paralelas

Aplicações paralelas computacionalmente intensivas freqüentemente utilizam dezenas de máquinas durante muitas horas. A falha de uma única máquina neste período normalmente faz com que toda a computação já realizada seja perdida. Deste modo, numa grade oportunista, onde máquinas ficam indisponíveis várias vezes em um único dia, a execução deste tipo de aplicação sem um mecanismo de tolerância a falhas é inviável.

Desenvolvemos um mecanismo de *recuperação por retrocesso baseada em checkpointing* [Elnozahy et al. 2002] que permite reiniciar uma execução interrompida de uma aplicação a partir do último *checkpoint* gerado [de Camargo et al. 2006c, de Camargo et al. 2006a]. Fornecemos suporte a aplicações seqüenciais, paralelas desacopladas (*bag-of-tasks*) e paralelas acopladas do tipo BSP (*Bulk Synchronous Parallel*). Um programa BSP é executado como uma seqüência de super-passos, onde cada super-passo é composto por uma fase de computação e uma de comunicação, que termina com uma barreira de sincronização. Uma vez que o modelo BSP já possui uma fase de sincronização, optamos por utilizar um protocolo de checkpointing coordenado [Elnozahy et al. 2002] para obter o estado global de uma aplicação BSP.

No mecanismo de *checkpointing* que implementamos, a aplicação é responsável por fornecer os dados que devem ser armazenados no *checkpoint* e por recuperar seu estado a partir dos dados presentes em um checkpoint [Bronevetsky et al. 2003]. Como a aplicação possui informação semântica sobre os dados que estão sendo armazenados ou recuperados, nosso mecanismo cria *checkpoints* portáveis, isto é, que podem ser gerados e recuperados em arquiteturas heterogêneas [de Camargo et al. 2005]. Finalmente, para que o programador não precise modificar o código-fonte de sua aplicação manualmente, desenvolvemos um pré-compilador baseado na ferramenta OpenC++ [Chiba 1995], que automaticamente analisa o código-fonte de uma aplicação C e o modifica de modo que esta armazene seu estado.

A geração do arquivo contendo o estado da aplicação é realizada pela biblioteca de *checkpointing*. Esta biblioteca também realiza a coordenação entre os processos de uma aplicação paralela no momento de gerar *checkpoints* globais consistentes, contendo o estado de todos os processos da aplicação. O armazenamento dos *checkpoints* é realizado pelo OppStore, descrito na Seção 3.

Desenvolvemos também um módulo gerenciador de execuções, denominado *Execution Manager* (EM), que monitora a execução de aplicações em um aglomerado InteGrade e, sempre que um dos processos de uma aplicação falha, este módulo inicia e coordena o processo de reinicialização daquela aplicação. Para tal, o EM notifica todos os processos da aplicação sobre a falha e fornece a estes processos a localização do último *checkpoint* armazenado. Estes processos então obtém este *checkpoint* e reiniciam sua execução a partir do estado nele contido.

3. Armazenamento Distribuído de Dados

Uma infra-estrutura que permita a execução de aplicações em máquinas não-dedicadas precisa também gerenciar os dados referentes a estas aplicações, sejam estes checkpoints, dados de entrada ou dados de saída. Além dos ciclos ociosos, as máquinas conectadas a uma grade oportunista normalmente possuem grandes quantidades de espaço livre em disco. Para permitir a utilização deste espaço livre para o armazenamento distribuído de dados, desenvolvemos o middleware OppStore [de Camargo and Kon 2007]. Nosso principal desafio foi como desenvolver um middleware que gerenciasse milhares de máquinas heterogêneas, utilizadas de modo oportunista e distribuídas geograficamente.

Optamos por organizar as máquinas da grade em uma federação de aglomerados, onde cada aglomerado é constituído por máquinas fisicamente próximas, por exemplo, em um mesmo laboratório ou departamento. Cada aglomerado contém uma máquina que

instancia um módulo que gerencia as máquinas daquele aglomerado, denominado CDRM (*Cluster Data Repository Manager*). As demais máquinas funcionam como repositórios de dados e instanciam o módulo ADR (*Autonomous Data Repository*).

Os CDRMs se comunicam através de uma rede *peer-to-peer*, utilizando o protocolo Pastry [Rowstron and Druschel 2001] como substrato. Cada CDRM possui um identificador único, que é utilizado para determinar os locais onde os arquivos serão armazenados. Os ADRs funcionam como repositórios de dados que recebem requisições para armazenamento e recuperação de dados em uma máquina. ADRs utilizam poucos recursos computacionais e podem ser configurados pelo dono da máquina para realizar transferências de dados somente quando a máquina está ociosa ou a qualquer momento.

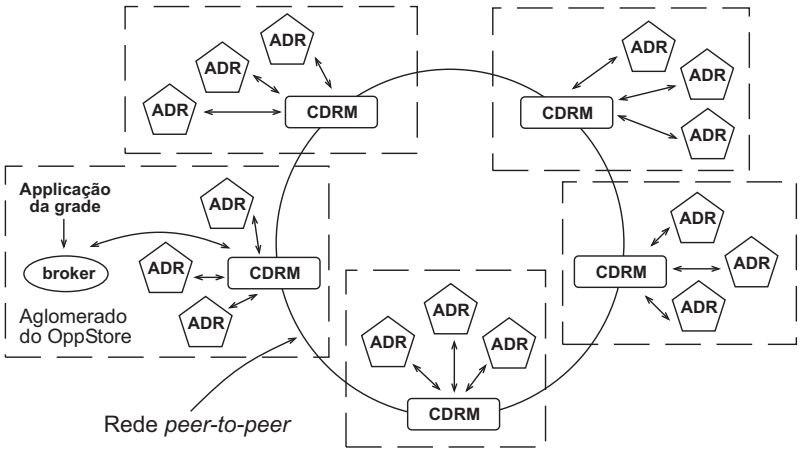

Figure 1. Arquitetura do OppStore.

Na Figura 1 podemos visualizar a arquitetura do OppStore e seus principais componentes. O sistema é organizado em dois níveis: (1) rede *peer-to-peer* composta pelos CDRMs e (2) aglomerados compostos por um CDRM e vários ADRs cada. Esta organização em dois níveis facilita o gerenciamento do dinamismo da grade, uma vez que as constantes mudanças de estado das máquinas podem ser tratadas internamente em cada aglomerado pelo CDRM.

Como os dados são armazenados em máquinas não-dedicadas, precisamos utilizar um mecanismo de redundância para garantir a disponibilidade destes dados. OppStore codifica arquivos que serão armazenados em fragmentos redundantes, utilizando um tipo de codificação de rasura (*erasure coding*) denominado IDA (*Information Dispersal Algorithm*)[Rabin 1989]. Nesta codificação, são gerados n fragmentos, sendo que quaisquer k fragmentos, onde $k < n$, são suficientes para reconstruir o arquivo original. Estudos analíticos mostram que, para um dado nível de redundância, dados armazenados com codificação de rasura possuem uma disponibilidade média várias vezes maior que utilizando replicação [Weatherspoon and Kubiatowicz 2002].

Aplicações e usuários da grade acessam o OppStore através de uma biblioteca denominada *access broker*, que funciona como um intermediador de acesso ao sistema e esconde do usuário os detalhes dos protocolos utilizados internamente pelo OppStore. Esta biblioteca possui uma interface que disponibiliza métodos que permitem o armazenamento e recuperação de arquivos. Diversos tipos de dados podem ser armazenados num sistema de grade, com cada tipo de dados possuindo diferentes requisitos. OppStore

permite que aplicações clientes escolham entre dois métodos de armazenamento para os arquivos: (1) armazenamento perene e (2) armazenamento efêmero.

No modo de armazenamento perene, os fragmentos codificados são distribuídos em diversos aglomerados da grade. Esta distribuição melhora a disponibilidade de dados armazenados, uma vez que estes podem ser recuperado mesmo quando aglomerados inteiros da grade ficam indisponíveis. Além disso, durante a recuperação de arquivos, aplicações podem obter fragmentos localizados nos aglomerados mais próximos, melhorando o desempenho e diminuindo o tráfego de dados na rede. Este modo é utilizado para armazenar arquivos de entrada e saída de aplicações.

Já no modo de armazenamento efêmero, os dados são armazenados apenas em máquinas do aglomerado de onde a requisição foi realizada. A vantagem deste modo é que os dados trafegam apenas pela rede local, gerando uma latência menor para o armazenamento e recuperação de dados. Este modo é utilizado, por exemplo, para armazenar *checkpoints* periódicos de aplicações.

Finalmente, as máquinas de uma grade computacional normalmente são heterogêneas e possuem diferentes padrões de utilização. É preferível armazenar fragmentos em máquinas que permanecem ociosas por longos períodos de tempo, possuem conexões de maior velocidade e maior quantidade de espaço livre em disco. Por outro lado, esta seleção de máquinas deve ser balanceada, de modo a não sobrecarregar algumas máquinas. Apesar de existirem técnicas para a distribuição de carga em redes *peer-to-peer*, sendo a principal delas o uso de servidores virtuais [Stoica et al. 2001], estas possuem limitações, como o alta sobrecarga gerada.

Para resolver este problema de heterogeneidade, propusemos o conceito de identificadores virtuais [de Camargo and Kon 2006], onde atribuímos a cada CDRM, além do identificador Pastry, um identificador virtual, que pode ser facilmente alterado para refletir a capacidade de cada aglomerado da grade. Conseqüentemente, podemos definir a probabilidade de que um aglomerado seja escolhido para armazenar um determinado fragmento em função de uma métrica de capacidade desejada.

4. Experimentos

Realizamos diversos experimentos para avaliar a sobrecarga do mecanismo de *checkpointing* sobre o tempo de execução de aplicações e o impacto do uso de diferentes estratégias de armazenamento sobre esta sobrecarga. Na Figura 2, apresentamos a sobrecarga do uso de *checkpointing* em uma aplicação paralela do tipo BSP que realiza uma seqüência de multiplicações de matrizes, utilizando diferentes estratégias de armazenamento e diferentes tamanhos de matrizes. Para cada caso, realizamos 16 execuções da aplicação e calculamos a média e o desvio padrão do tempo de execução. O eixo x contém 5 estratégias de armazenamento: (1) execução sem *checkpointing*, (2) armazenamento de 2 réplicas do *checkpoint*, (3) codificação utilizando paridade, (4) codificação com IDA gerando 8 fragmentos, dos quais 7 são necessários para a recuperação do arquivo e (5) codificação com IDA gerando 8 fragmentos, dos quais 6 são necessários. O eixo y contém o tempo de execução médio para cada cenário e as barras representam o desvio padrão. Os valores $nCkp = ...$ representam o número médio de *checkpoints* gerados durante a execução da aplicação em cada cenário.

Os resultados mostram que a utilização de IDA (cenários 4 e 5) causa a maior

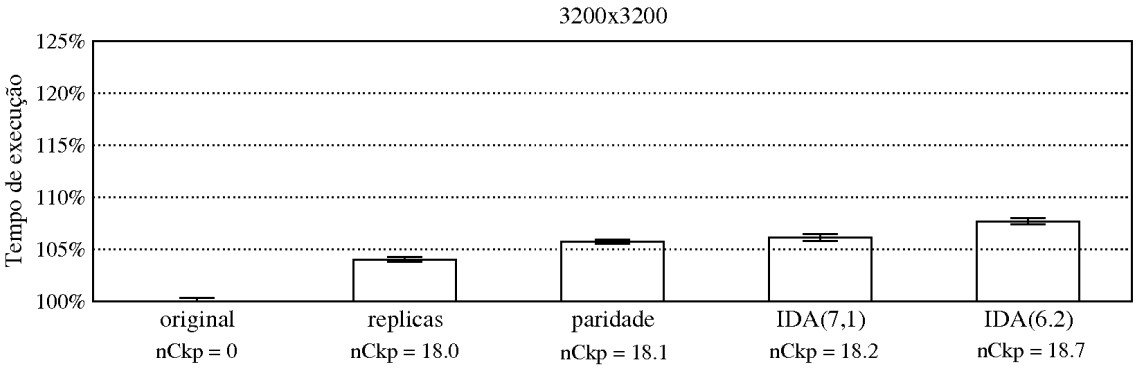

Figure 2. Sobrecarga do armazenamento de *checkpoints* sobre o tempo de execução da aplicação de multiplicação de matrizes.

sobrecarga. Isto era esperado, dado que é necessário realizar a codificação dos dados. No caso de matrizes de tamanho 3200x3200, onde são gerados *checkpoints* globais de 351.6MB com um intervalo mínimo de $60s$, a sobrecarga é de aproximadamente 7.5% para a estratégia *IDA(m=6,k=2)*. Podemos facilmente reduzir esta sobrecarga para valores menores que 2% aumentando o intervalo entre *checkpoints* para 5 minutos. Deste modo, vemos que mesmo para uma aplicação que gera *checkpoints* de tamanhos elevados, a sobrecarga do mecanismo é bastante baixa.

Avaliamos o OppStore utilizando simulações e experimentos. No primeiro caso, simulamos uma grade oportunista em condições realistas, utilizando padrões de uso obtidos em medições de máquinas reais, que mostram que estas máquinas permanecem ociosas entre 25% e 80% do tempo, dependendo do aglomerado e do período do dia. Simulamos o procedimento de armazenamento para 10 mil arquivos, para então realizar sua recuperação, com o objetivo de avaliar a quantidade de arquivos que conseguimos recuperar considerando que podemos obter apenas fragmentos de máquinas ociosas.

A partir de simulações, mostramos que, utilizando apenas os períodos ociosos de máquinas compartilhadas para armazenar e recuperar dados, podemos obter disponibilidades de dados de 99.9% utilizando um fator 3 de replicação e 93.2% utilizando um fator 2 de replicação [de Camargo and Kon 2007]. Estes excelentes índices de disponibilidade de dados foram obtidos em função do uso de identificadores virtuais para realizar a escolha dos locais de armazenamento de fragmentos.

Realizamos experimentos com o OppStore em um ambiente real de uma grade oportunista composta por cinco aglomerados, três em São Paulo (sp1, sp2 e sp3), um em Goiânia (go) e um em São Luiz (sl), conectados pela Internet pública. Goiânia e São Luiz estão distantes 900km e 3000km de São Paulo, respectivamente. O *access broker* foi instanciado em um computador Athlon64 de 2GHz com 4GB de memória RAM e sistema operacional Linux 2.6, localizado no aglomerado sp1, em São Paulo.

Realizamos o armazenamento e recuperação de arquivos com tamanhos entre 100KB e 500MB. Configuramos o *access broker* para codificar o arquivo em 5 fragmentos, sendo 2 suficientes para reconstrui-lo. O gráfico à esquerda na Figura 3 mostra o tempo necessário para finalizar o processo de codificação dos dados (linha pontilhada) e para finalizar o processo de armazenamento (linha contínua), com relação ao tamanho

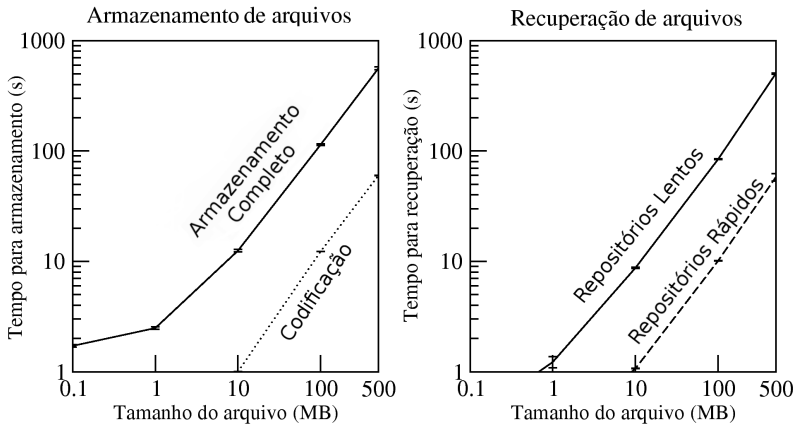

Figure 3. Armazenamento e recuperação de dados armazenados.

do arquivo. No caso de um arquivo de 500MB, o *access broker* utilizou 60 segundos para codificar o arquivo e de 560 segundos para transferir os 5 fragmentos de 500MB gerados. O tempo de armazenamento é limitado pela conexão mais lenta, neste caso a conexão entre São Paulo e Goiânia, com uma taxa de transferência média de 400KB/s. Já para a recuperação, basta recuperar 2 dos 5 fragmentos armazenados. Na Figura 3, o gráfico à direita mostra o tempo necessário para recuperar um arquivo utilizando os dois repositórios com as conexões mais rápidas (linha tracejada) e mais lentas (linha contínua), com relação ao tamanho do arquivo. Utilizando os servidores mais rápidos, foram necessários apenas 58 segundos para obter os fragmentos e reconstruir o arquivo.

5. Conclusões

Grades oportunistas, como o InteGrade, permitem o acesso a grandes quantidades de recursos computacionais utilizando máquinas já presentes nas instituições. Estas grades são particularmente importantes no cenário brasileiro, onde universidades e centros de pesquisa normalmente possuem poucos recursos humanos, financeiros e de espaço físico.

A infra-estrutura computacional que desenvolvemos neste trabalho permite a utilização destas máquinas de modo eficiente e tolerante a falhas através dos middlewares InteGrade e OppStore. Os resultados experimentais indicam que OppStore provê uma plataforma viável e de baixo custo para resolver o problema do armazenamento de dados em grades computacionais oportunistas.

Durante o desenvolvimento do trabalho realizamos diversas publicações, que nos permitiram divulgar o trabalho desenvolvido e obter valiosas sugestões que permitiram o seu aprimoramento. Como trabalho em andamento, estamos continuando o desenvolvimento do middleware OppStore e realizando experimentos em redes de grande escala compostas por centenas de máquinas. Pretendemos implantar e monitorar o uso do OppStore sobre o InteGrade em uma grade de grande área de modo a verificar seus padrões de uso e assim, aprimorar os seus protocolos.

References

Bronevetsky, G., Marques, D., Pingali, K., and Stodghill, P. (2003). Automated application-level checkpointing of MPI programs. In *PPoPP '03: 9th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 84–89.

Chiba, S. (1995). A metaobject protocol for C++. In *OOPSLA '95: Proceedings of the 10th ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 285–299.

de Camargo, R. Y., Cerqueira, R., and Kon, F. (2006a). Strategies for checkpoint storage on opportunistic grids. *IEEE Distributed Systems Online*, 18(6).

de Camargo, R. Y., Goldchleger, A., Carneiro, M., and Kon, F. (2006b). The Grid architectural pattern: Leveraging distributed processing capabilities. In *Pattern Languages of Program Design 5*, pages 337–356. Addison-Wesley Publishing Company.

de Camargo, R. Y., Goldchleger, A., Kon, F., and Goldman, A. (2006c). Checkpointing BSP parallel applications on the InteGrade Grid middleware. *Concurrency and Computation: Practice and Experience*, 18(6):567–579.

de Camargo, R. Y. and Kon, F. (2006). Distributed data storage for opportunistic grids. In *MDS '06: Proceedings of the 3rd ACM/IFIP/USENIX International Middleware Doctoral Symposium*, Melbourne, Australia.

de Camargo, R. Y. and Kon, F. (2007). Design and implementation of a middleware for data storage in opportunistic grids. In *CCGrid '07: Proc. of the 7th IEEE/ACM Int. Symposium on Cluster Computing and the Grid*, Rio de Janeiro, Brazil.

de Camargo, R. Y., Kon, F., and Goldman, A. (2005). Portable checkpointing and communication for BSP applications on dynamic heterogeneous Grid environments. In *SBAC-PAD'05: The 17th International Symposium on Computer Architecture and High Performance Computing*, pages 226–233, Rio de Janeiro, Brazil.

Elnozahy, M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Comp. Surveys*, 34(3):375–408.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004). InteGrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16:449–459.

Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348.

Rowstron, A. I. T. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: IFIP/ACM Int. Conf. on Distributed Systems Platforms*, pages 329–350, Heidelberg, Germany.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: The 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160.

Thain, D., Tannenbaum, T., and Livny, M. (2002). Condor and the grid. In Berman, F., Fox, G., and Hey, T., editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc.

Weatherspoon, H. and Kubiatowicz, J. (2002). Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 328–338, London, UK. Springer-Verlag.