Biblioteca de Comunicação Coletiva para Ambientes Distribuídos Dinâmicos

Viviane Thomé e Lúcia Drummond
Universidade Federal Fluminense
Instituto de Computação / IC-UFF
Niterói, Rio de Janeiro, Brasil
{vthome, lucia}@ic.uff.br

Resumo

Usualmente, sistemas distribuídos apresentam características dinâmicas, tais como variações no desempenho, falhas e recuperações dos canais de comunicação. Existem vários trabalhos que propõem a utilização de uma árvore geradora para a realização de operações coletivas em ambientes distribuídos. Na maior parte deles, a criação desta topologia ocorre no início da execução da aplicação e não considera posteriores alterações no ambiente de execução. Este trabalho apresenta uma ferramenta que disponibiliza operações coletivas para o MPI, considerando características dinâmicas do sistema. Para isso, além da construção inicial de uma árvore geradora de custo mínimo para a representação da topologia, a ferramenta também realiza a sua constante adaptação, através de dados coletados pelo NWS - Network Weather Service. Tanto a geração como a adaptação da árvore geradora de custo mínimo são realizadas através de algoritmos distribuídos.

1. Introdução

A maior parte das ferramentas de programação paralela atuais disponibiliza operações de comunicação em grupo baseadas em estruturas estáticas pré-definidas, tais como: árvores binomiais, árvores que distinguem canais pertencentes ao mesmo *site* dos canais que interconectam *sites* distantes ou, também, árvores organizadas conforme uma hierarquia multinível. Tais operações não consideram características dinâmicas usualmente apresentadas por sistemas distribuídos durante a execução de uma aplicação, como alteração no desempenho, falha e recuperação dos canais de comunicação e processadores.

Este trabalho apresenta uma ferramenta para o ambiente MPI que fornece operações coletivas que consideram mudanças significativas de desempenho em um ou mais canais de comunicação. A topologia usada para tais operações é representada por uma Árvore Geradora de custo Mínimo (AGM), construída e adaptada dinamicamente através de algoritmos distribuídos, o que dispensa a concentração de informações sobre a rede em um único nó central. Para que a AGM reflita, durante a execução da aplicação, as reais condições do ambiente, é necessário monitorar a rede. Caso sejam verificadas mudanças, em relação às condições anteriores, a árvore é atualizada. Cada processo monitora os seus canais adjacentes através da ferramenta Network Weather Service [11, 13, 14, 15].

Este trabalho se diferencia dos demais que tratam de operações de comunicação coletiva [2, 12, 4, 10, 7, 9] em relação, principalmente, às seguintes propostas:

- Utilização de uma árvore geradora de custo mínimo, construída de forma distribuída;

- Adaptação da árvore geradora mínima durante a execução da aplicação para refletir eventuais mudanças ocorridas no ambiente;

- Utilização de dados coletados pelo NWS para criação e adaptação da árvore geradora mínima.

Testes foram realizados para avaliar o desempenho das operações de comunicação coletiva da ferramenta proposta em relação às empregadas mais comumente nas implementações do padrão MPI. Constatamos que a ferramenta proposta permitiu reduzir o tempo de execução dessas operações significativamente, considerando especialmente ambientes com diversas variações de desempenho de canais de comunicação.

O restante deste artigo está organizado da seguinte forma. Na seção seguinte são apresentados os trabalhos relacionados, com suas propostas e principais diferenças em relação ao nosso trabalho. A Seção 3 descreve os algoritmos distribuídos utilizados para a construção e adaptação da AGM. Na Seção 4, apresentamos detalhes sobre a nova biblioteca MPI e sua integração com o NWS. Os resultados dos experimentos computacionais são apresentados na Seção 5. Finalmente, na Seção 6 estão as conclusões.

2. Trabalhos Relacionados

Existem vários trabalhos recentes que tratam de operações de comunicação coletiva. Saito e Taura [12] propõem um método para execução de operações coletivas em árvores criadas dinamicamente, em tempo de execução, de acordo com o conhecimento da topologia existente. Este trabalho apresenta as seguintes diferenças em relação ao aqui proposto: as árvores usadas não são de custo mínimo, a biblioteca usada para troca de mensagens neste trabalho é a Phoenix e a monitoração da rede é feita através de *pings*, que não fornece informação sobre o desempenho do canal de comunicação no nível da aplicação.

Burger et al. apresentam a ferramenta TopoMon em [2], que utiliza um processo central para reunir informação sobre a rota entre todos os sítios de um ambiente de Grade e gerar a topologia a ser utilizada pelas aplicações do usuário e bibliotecas de comunicação. O nó central cria dois tipos de árvores: uma de latência mínima e outra de largura de banda máxima. Para previsão de latência também usa-se o NWS . Note que neste trabalho usa-se o algoritmo seqüencial de Dijkstra para determinação dos caminhos mínimos e toda informação sobre a topologia é concentrada em um único nó.

As implementações do padrão MPI também evoluíram em relação às operações coletivas. Tipicamente, o MPI monta uma árvore binomial para comunicações coletivas sem levar em consideração a localização dos processos. Mais recentemente alguns trabalhos foram propostos objetivando melhorar as estruturas usadas para as operações de comunicação coletiva [4, 5, 6, 10, 7, 8, 9]. Nas implementações MPI-StarT [4], MagPIe [7] e MPI-LAM [8] a rede é vista em duas camadas. MPI-StarT distingue entre comunicação intra e interclusters, enquanto MagPIe e MPI-LAM diferenciam comunicação entre LAN e WAN.

Karonis et al [10] apresentam uma implementação de operações coletivas no MPICH-G baseada na visão multicamada da rede obtida através de informações disponibilizadas pelo Globus. Este trabalho foi aperfeiçoado em [5, 6, 9] permitindo melhoria na eficiência da execução de operações coletivas em ambientes de Grade. Mais especificamente, na biblioteca MPICH-G2 [9], cada processo é classificado de acordo com a sua localização. Esta classificação é realizada no início da execução da aplicação através da construção de tabelas, chamadas de Tabelas de Cores e de Identificação de Clusters, e se mantêm constantes durante toda a execução da aplicação.

Todos estes trabalhos para MPI utilizam uma topologia estática.

3. Algoritmos Distribuídos para Geração e Adaptação da Árvore Geradora de Custo Mínimo

Uma árvore geradora de uma rede representa uma estrutura conectada contendo todos os nós desta rede e no contexto deste trabalho é empregada em redes ponto-a-ponto para disseminação eficiente de mensagens.

Foram implementados dois algoritmos distribuídos, para esta ferramenta: um para construção e outro para adaptação da árvore geradora mínima (AGM).

O algoritmo implementado para a criação da árvore inicial se baseia no GHS [3]. O GHS é um algoritmo distribuído assíncrono que determina a árvore geradora mínima sobre um grafo. Cada nó do grafo é um processo que sabe inicialmente os pesos dos canais adjacentes. Os processos executam o mesmo algoritmo e trocam mensagens com seus vizinhos até que a árvore seja construída. Depois que cada processo termina seu algoritmo local, ele sabe quais canais adjacentes estão presentes na árvore. O algoritmo se baseia no conceito de fragmento, que é uma subárvore da árvore geradora final.

Inicialmente todos os nós são fragmentos. No decorrer do algoritmo eles se unem em fragmentos maiores até se formar a AGM. Durante a criação da árvore, cada processo participante de um fragmento poderá estar em um dos seguintes estados: *sleeping* - não está participando do algoritmo, *find* - está procurando a aresta de custo mínimo a ser incluída na árvore e *found* - encontrou a aresta de custo mínimo.

Em relação à classificação dos canais adjacentes a cada processo, inicialmente todos estão no estado *basic* - indicando que ainda não foi analisado. Após o canal ser incluído na árvore, este é classificado como *branch* e, no caso de ser impedida a sua participação na árvore, devido à formação de ciclos, o canal torna-se *rejected*.

O desenvolvimento do algoritmo para adaptação da árvore geradora mínima distribuída baseou-se na proposta do artigo [1]. O objetivo deste algoritmo é realizar a atualização da árvore geradora mínima em uma rede com mudanças topológicas, sem precisar re-executar o algoritmo GHS. Esse algoritmo pode responder a múltiplas falhas e recuperações de canais. Neste trabalho consideramos como falha uma significativa piora no desempenho do canal e, reciprocamente, como recuperação uma sensível melhoria no desempenho do mesmo. Dado um número finito de mudanças topológicas ocorridas durante um período, o algoritmo encontra a árvore geradora mínima correspondente às últimas condições da rede. Optamos por tratar primeiro todas as falhas e, posteriormente, as recuperações.

No caso de ocorrência de falhas, a árvore inicial estará dividida em dois ou mais fragmentos (subárvores de custo mínimo). Da mesma forma como foi tratado no algoritmo

GHS, os fragmentos serão reunidos até formar um único que corresponderá à AGM. Em relação às recuperações de canais, o algoritmo de adaptação é capaz de detectar, caso exista, um ciclo contendo o canal recuperado, o algoritmo decide qual canal deverá ser retirado da árvore. Naturalmente, o canal a ser retirado será aquele que possuir o maior valor de latência no ciclo formado.

São acrescentados dois novos estados possíveis, para um fragmento, em relação ao algoritmo GHS, durante a execução da adaptação da árvore: *reiden* - significa que o fragmento está tratando a ocorrência de uma falha e *recover* - o tratamento está sendo feito para a recuperação de um canal. Em relação aos canais adjacentes, a classificação feita no GHS também é utilizada aqui.

Como contribuição do trabalho aqui proposto, foram incluídos neste algoritmo dois procedimentos de terminação. O primeiro refere-se à detecção de terminação do tratamento de falhas, para que o algoritmo passe a tratar as recuperações. O segundo procedimento trata da detecção de terminação das recuperações e, conseqüentemente, a identificação do fim do processo de adaptação da AGM. Estes procedimentos são necessários para garantir que uma rodada de ajustes tenha terminado antes de se iniciar a outra e a execução da aplicação somente continue depois que todos os ajustes tenham sido concluídos.

A seguir serão detalhadas as implementações dos procedimentos para detecção do fim do tratamento de falhas e recuperações, respectivamente.

Cada processo mantém uma variável local *tag*, inicializada com valor zero, para contabilizar em que rodada do tratamento de terminação de falhas está e um vetor de vizinhos. Neste vetor são armazenadas informações sobre qual estágio, durante o processo de terminação de falhas, cada vizinho se encontra.

São trocados dois tipos de mensagens: *not_fail* e *not_fail_ack*. Se um processo, no início do procedimento de tratamento de falhas, identificar que não possui falhas a tratar, incrementa a variável *tag* e envia para todos os seus vizinhos, na árvore AGM, a mensagem *not_fail (tag)*.

Um processo ao receber a mensagem *not_fail (tag)* compara o valor da *tag* recebida com a sua variável local. Caso o valor da *tag* seja maior, significa que uma nova suspeita de terminação de falhas está se iniciando, logo, o processo armazena na sua variável local o valor da *tag* recebida. Se o processo não estiver participando de nenhum tratamento de falhas, o próximo passo é repassar a mensagem *not_fail (tag)* para os demais vizinhos na árvore. Caso seja um processo folha, a mensagem *not_fail_ack (tag)* será enviada ao processo pai.

Quando um processo recebe a mensagem *not_fail_ack (tag)*, ele verifica se a *tag* recebida é igual a sua variável local. Se os valores forem diferentes a mensagem é ignorada, senão, o processo inclui no seu vetor, na posição correspondente ao vizinho, o valor da *tag* recebida. Em seguida, o processo percorre este vetor para verificar se todos os seus processos filhos já lhe enviaram a mensagem *not_fail_ack (tag)*. Se isto for verdade, o processo analisa se é a raiz da árvore, neste caso, a mensagem de fim de tratamento de falhas *gosleep_fail* é disseminada por toda a árvore, caso contrário, a mensagem *not_fail_ack (tag)* é repassada para o processo pai. Cada processo ao receber *gosleep_fail* identifica que pode ser iniciada o tratamento das recuperações, pois, o seu tratamento de falhas já se encerrou.

Em relação ao tratamento de terminação das recuperações, são trocadas mensagens do tipo *not_rec (tag)* e *not_rec_ack (tag)*. Um processo ao suspeitar do término de tratamento de recuperações envia mensagens *not_rec (tag)* para vizinhos e, no caso de ser processo folha ou já ter recebido de todos os filhos a mensagem *not_rec_ack (tag)*, envia *not_rec_ack (tag)* para o processo pai. Aqui empregamos um procedimento semelhante ao adotado na terminação das falhas para detectar o fim do tratamento das recuperações. Logo, o processo raiz ao receber *not_rec_ack (tag)* de todos os seus filhos, envia pela árvore a mensagem *gosleep_rec*, indicando que todos os ajustes na árvore foram concluídos, permitindo que a execução da aplicação prossiga sem problemas.

4. Implementação da Biblioteca de Comunicação Coletiva Dinâmica

4.1 Ferramenta de Monitoração NWS

O NWS opera um conjunto distribuído de sensores que reúne informações instantâneas sobre os recursos [11, 13, 14, 15]. Trata-se de uma ferramenta que executa monitoração de forma distribuída para produzir previsões dinâmicas de desempenho dos recursos levando em consideração as medidas armazenadas.

É possível monitorar os seguintes recursos através do NWS: a fração de CPU disponível para os processos; a quantidade de espaço disponível em disco; a quantidade de memória livre disponível na máquina; o tempo requerido para estabelecer uma conexão TCP e a latência e banda TCP entre pares de processos.

A latência é obtida pelo NWS através da medição do RTT (Round-Trip Time) de um pacote. O RTT é o tempo gasto (tempo de ida e volta), por um pacote pequeno, para viajar do cliente ao servidor. A latência poderia ser medida pelo próprio ping, mas poderia ocorrer filtragem de pacotes.

O papel do NWS, neste trabalho, é fornecer as medidas de latência entre todos os processos, para que se possa construir e manter atualizada uma árvore geradora mínima, a ser usada pela nossa biblioteca MPI para operações coletivas.

4.2 Visão Geral da Biblioteca

A biblioteca MPI proposta possui as seguintes operações implementadas: *MPI_Init*, *MPI_Init_thread*, *MPI_Bcast* e *MPI_Finalize*. Estas funções poderão ser chamadas pelo programador sem a necessidade de mudança na assinatura de nenhuma delas. Isto acontece porque a nossa biblioteca, ao verificar a existência da chamada de uma dessas funções na aplicação do usuário, substitui a operação MPI original pela nossa implementação, sendo tudo realizado de forma transparente.

Atualmente, apenas a operação coletiva *MPI_Bcast* foi implementada, mas futuramente as demais operações de comunicação coletiva serão reescritas.

A seguir serão apresentadas sucintamente as modificações realizadas nas operações MPI para se adequar às necessidades da nossa ferramenta, que realiza a disseminação de mensagens através de uma AGM.

A função *MPI_Init* da biblioteca MPI proposta possui as mesmas funcionalidades presentes na operação *MPI_Init* original. Entretanto, na nossa implementação foram alocadas estruturas para armazenamento das informações necessárias para a construção e armazenamento da AGM. Ainda nesta operação, é realizada a chamada à função que constrói a AGM através do algoritmo distribuído proposto em [3]. Após a árvore ter sido montada, a operação coletiva *MPI_Bcast* poderá ser executada.

Para *MPI_Init_thread* todas as considerações feitas na construção da função *MPI_Init* são válidas para a implementação desta operação.

A implementação da nossa operação *MPI_Bcast* foi feita através das operações ponto-a-ponto *MPI_Send* e *MPI_Recv*. Os seguintes procedimentos foram realizados para a sua execução:
(i) inicialmente o procedimento para a adaptação da AGM é chamado, neste momento realizamos a integração da nossa biblioteca com a ferramenta NWS. Neste procedimento são obtidos os valores das latências instantâneas dos canais adjacentes ao processo através da consulta ao NWS;
(ii) de posse dos valores das latências obtidos pela consulta, é verificado se algum canal sofreu alteração. Desta forma, a biblioteca classifica as variações como ocorrências de falhas e/ou recuperações de canais;
(iii) o próximo passo, caso tenham sido identificadas alterações nos canais, é a realização da adaptação da árvore através do algoritmo distribuído [1];
(iv) em seguida, é feita a disseminação da mensagem pela AGM, que pode ter sido adaptada ou não, através das operações *MPI_Send* e *MPI_Recv*.

A última operação MPI implementada por esta biblioteca foi a função MPI_Finalize que, como acontece na operação MPI_Init, realiza as mesmas atividades que a operação original, mas tendo como atividade adicional a preocupação em liberar as estruturas de dados utilizadas para armazenamento e adaptação das informações da árvore geradora mínima.

Para as operações MPI que sejam chamadas pela aplicação e não tenham sido implementadas pela nossa biblioteca, a aplicação continuará a utilizar as operações originais, uma vez que a nossa biblioteca implementada referencia a biblioteca MPI original.

5. Resultados Computacionais

Para a realização dos testes, foram utilizadas 24 máquinas com processador Pentium IV, 2.6 GHz de freqüência, 512 Mb de memória RAM, sistema operacional GNU/Linux versão 2.6.8-1.521, biblioteca MPI-LAM-7.0.6 e compilador versão i386-redhat-linux/3.3.3. As máquinas, pertencentes a uma mesma rede, encontravam-se dedicadas exclusivamente aos testes, e, para simular um ambiente heterogêneo, foi definida uma topologia constituída de 6 *sites* com 4 máquinas cada, onde um *site* representa uma localidade geográfica distinta e todos os processos se comunicam por canais com valores de latências variados.

Com o objetivo de obter valores mais próximos à realidade, as latências entre os *sites* foram estimadas com base em *pings* realizados a universidades do Brasil, China e Estados Unidos.

Os *sites* foram identificados pelos nomes S0, S1, S2, S3, S4 e S5 e a distribuição foi realizada da seguinte forma: consideramos três *sites* localizados no Brasil (S0, S3 e S5), dois nos Estados Unidos (S1 e S4) e um na China (S2).

A definição da topologia, descrita acima, foi adotada em todos os testes realizados e pode ser vista na Tabela 1, onde também são apresentados as identificações dos processos que compõem cada *site* e os valores das latências iniciais em *ms*. As latências entre processos do mesmo *site* foram consideradas com valores próximos a zero.

Tabela 1. Latências entre os sites.

	S0 00-03	**S1 04-07**	**S2 08-11**	**S3 12-15**	**S4 16-19**	**S5 20-23**
S0	0.0	485.4	698.9	14.9	332.8	61.4
S1	485.4	0.00	364.1	583.8	13.5	490.5
S2	698.9	364.1	0.0	701.2	371.7	722.9
S3	14.9	583.8	701.2	0.0	331.0	35.1
S4	332.8	13.5	371.7	331.0	0.0	355.9
S5	61.4	490.5	722.9	35.1	355.9	0.0

A eficiência da proposta aqui apresentada foi avaliada comparando o desempenho dos algoritmos implementados *MPICH-like*, *MagPIe-like* e AGM. Os dois primei-

ros referem-se às implementações de comunicação coletiva conforme proposto em *MPICH* e *MagPIe*, que não se encontravam disponíveis no ambiente utilizado. No AGM a análise foi feita através de execuções com e sem a atualização da árvore gerada inicialmente. Como observamos pequenas variações de tempo em execuções sucessivas de uma mesma aplicação, cada um dos testes foi executado três vezes. As figuras e tabelas seguintes apresentam estas médias.

A primeira seqüência de testes comparou o desempenho dos algoritmos *MPICH-like* e *MagPIe-like*, onde foi possível verificar a vantagem em se utilizar a árvore gerada pela versão *MagPIe-like* que detém o conhecimento da topologia em duas camadas. *MagPIe-like* consegue identificar quais processos pertencem ao seu *site* e quais estão mais distantes fisicamente, enquanto o algoritmo *MPICH-like* constrói uma árvore binomial sem a distinção da localização dos processos. O algoritmo *MagPIe-like* tenta minimizar a quantidade de comunicação entre processos de *sites* diferentes da raiz do *broadcast* até o seu destino.

As Figuras 1 e 2 representam, respectivamente, as árvores geradas pelos algoritmos *MPICH-like* e *MagPIe-like*, que possuem como raiz do *broadcast* o processo 12. Além disso, também mostram os *sites* a que os nós pertencem. A Tabela 2 demonstra o tempo total, em segundos, da aplicação com a variação da quantidade de MPI_Bcast's e a porcentagem de melhoria do *MagPIe-like* em relação ao *MPICH-like*.

Tabela 2. Execuções MPICH-like e MagPIe-like com tempos em segundos.

No de bcast's	1	4	16
MPICH-like	4.93	6,95	15.88
MagPIe-like	3.80	5,40	13.08
Melhoria	22,92%	22,30%	17,63%

Neste teste, já é possível observar que a utilização da árvore gerada pelo algoritmo *MagPIe-like* para a realização do *broadcast*, tende a ser melhor. Pode-se observar que o tempo de espera para o recebimento da mensagem pelos processos (0) e (8) tende a diminuir com o *MagPIe-like*, uma vez que pelo algoritmo *MPICH-like*, estes processos deveriam esperar duas comunicações entre *sites*. Para uma mensagem alcançar o processo (0), através do *MPICH-like*, primeiro a raiz do *broadcast* (12) deverá enviar a mensagem para processo (20) e este, em seguida, para o processo (0), o que levará aproximadamente 96,5 *ms*. Com o *MagPIe-like* o processo (0) deverá aguardar apenas 14,9 *ms* para o recebimento desta mensagem, um ganho equivalente a 18% em relação ao tempo de espera. O algoritmo *MagPIe-like* foi escolhido para comparações nos testes apresentados

Figura 1. Árvore gerada pelo algoritmo MPICH-like.

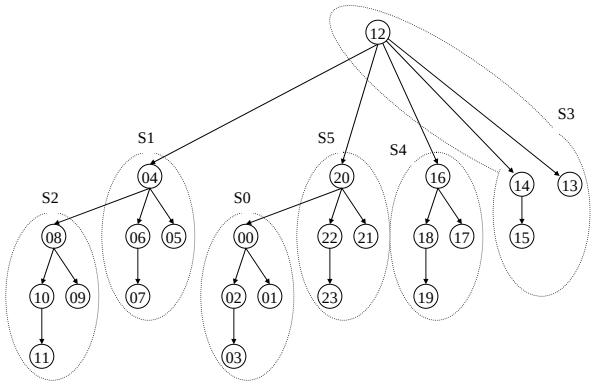
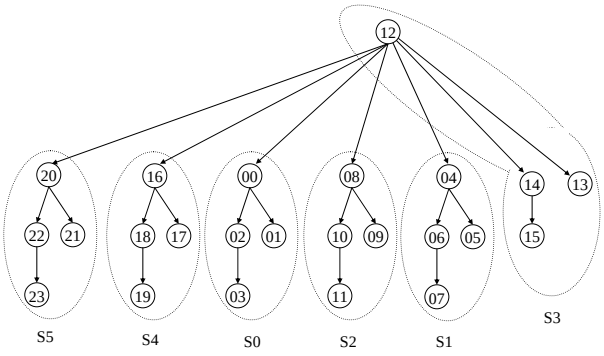


Figura 2. Árvore gerada pelo algoritmo MagPIe-like.

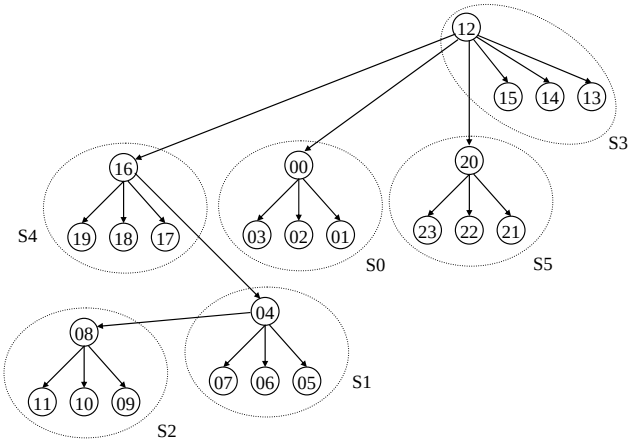


a seguir, devido ao seu melhor desempenho. Na figura 3, pode ser vista a árvore criada pelo algoritmo AGM e na figura 4 são apresentados os gráficos para testes realizados com a execução de 4, 8 e 16 *broadcasts* consecutivos com mensagens de tamanhos iguais a 24 *bytes*. Nestes gráficos são comparadas as médias obtidas em três execuções dos tempos de geração da árvore, tempo total da aplicação e a diferença entre os tempos gastos em toda a aplicação e na geração da árvore.

Importante notar que o algoritmo AGM, nestas execuções, não apresentou atualização da árvore e nenhuma latência sofreu modificação no seu valor. O objetivo destes testes foi mostrar o custo associado à criação desta estrutura para disseminar a informação e, apontar a partir de que momento a sua utilização torna-se vantajosa.

Analisando os gráficos da figura 4, é fácil ver que a versão *MagPIe-like* mostrou-se mais rápida, em relação ao tempo total da aplicação, em todas as execuções. No entanto, comparando os tempos de criação da árvore em relação ao tempo total da aplicação verificou-se o seguinte:

Figura 3. Árvore gerada pelo algoritmo AGM.



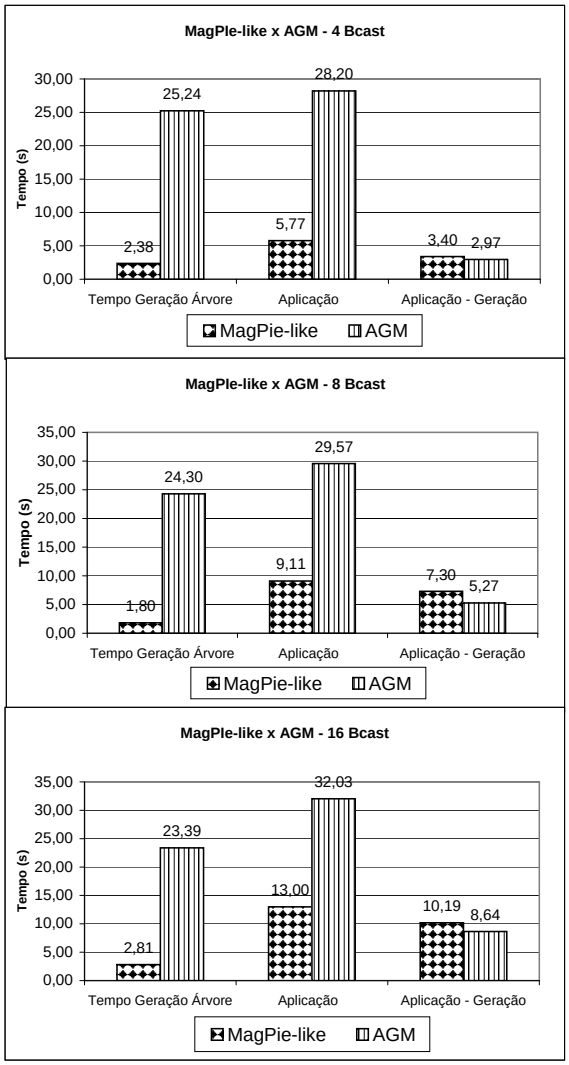
o algoritmo *MagPIe-like* utilizou 41%, 20% e 22% do tempo total da aplicação para a criação da árvore para as execuções com 4, 8 e 16 *broadcasts*, nesta ordem, sendo o restante do tempo usado para a disseminação da informação. Por outro lado, a AGM possui um custo maior na construção da árvore: 90%, 82% e 73% para 4, 8 e 16 *broadcasts*.

O terceiro conjunto de barras dos gráficos, "Aplicação - Geração", representa o tempo destinado à disseminação da informação através das operações MPI_Bcast, observa-se que para todos os três casos a AGM teve um desempenho melhor que *MagPIe-like*. Para as execuções com 4, 8 e 16 *broadcasts* consecutivos a AGM obteve uma melhoria de 13%, 28% e 15% em seus tempos, respectivamente. Portanto, pela análise deste conjunto de barras, é possível ver que a árvore utilizada pelo algoritmo AGM para a realização do *broadcast* se mostra mais eficiente.

A utilização da AGM irá se tornar mais vantajosa à medida que as latências entre os processos das árvores estáticas, geradas pelo algoritmo *MagPIe-like*, sofrerem variações mais significativas e a quantidade de MPI_Bcast's for incrementada na aplicação. Esta observação será confirmada nos testes apresentados mais adiante. Outra observação importante, é que AGM pode ter mais de uma comunicação entre *sites* durante a difusão das mensagens até alcançar um destino, como ocorre com o *MPICH-like*. Porém, é importante lembrar que isto ocorre somente quando o custo total é minimizado, o que não é garantido pelo *MPICH-like*.

O algoritmo distribuído que constrói a AGM pode ser executado de duas formas: com e sem adaptação da árvore geradora mínima após a definição inicial da árvore. Caso a opção de execução seja com adaptação da árvore, a cada execução da operação MPI_Bcast, é avaliado se há necessidade de atualização da árvore geradora mínima. Nos testes,

Figura 4. Comparação entre os algoritmos MagPIe-like e AGM.



esses algoritmos foram identificados como AGM e AGM-adap, o primeiro somente constrói a árvore e o segundo, além da construção, realiza a manutenção da árvore durante toda a aplicação.

Um fator importante, na versão AGM-adap, é que antes de cada execução da operação MPI_Bcast sempre são coletados, pelo NWS, os últimos valores de latências, para identificar possíveis mudanças nos canais.

Sempre que houver variações nos valores das latências, estas serão classificadas como uma falha ou uma recuperação, o que provocará a atualização da árvore. Nem sempre este procedimento será vantajoso, principalmente, se a variação de latência for muito pequena, pois, a árvore atualizada pode ser mais custosa do que a sem adaptação.

Para minimizar este esforço de atualização desnecessário, foi incluída uma opção de execução, que permite definir a partir de qual porcentagem da variação do valor da latência, será realizada a atualização da árvore.

Outra desvantagem é que o procedimento de verificação dos valores das latências pode executar desnecessariamente, quando não ocorrem mudanças nas latências dos canais, adicionando um custo na aplicação, mesmo sem nenhuma adaptação na árvore. Embora, o tempo para a verificação seja relativamente pequeno, 0.57 segundos. Para minimizar o custo de análise dos valores das latências, foi disponibilizada a opção que define o intervalo da quantidade de *broadcasts* para que seja feita esta análise. As execuções que possuem esta opção são identificadas por AGM-adapII. Testes mostraram as vantagens destas novas abordagens, como pode ser observado nas figuras 5, 6 e 7. Os tempos de execução apresentados em cada coluna são os maiores valores obtidos entre todos os processos.

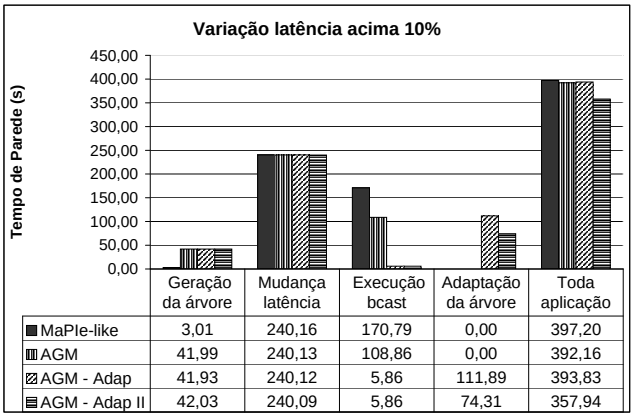
Nas figuras 5 e 6 foram realizadas execuções com 16 *broadcasts* consecutivos, tendo como parâmetros os valores 10% e 70% para a realização da adaptação da árvore. Novos valores nas latências foram inseridos após a geração da árvore inicial e, imediatamente, antes da execução da primeira operação MPI_Bcast. A inserção destes valores, artificialmente, provocou um acréscimo no tempo de execução da aplicação, que foi referenciado nestas figuras como "Mudança latência".

Analisando o gráfico da Figura 5, verifica-se que o tempo de geração da árvore inicial, como era de se esperar, na versão *MagPIe-like* mostrou-se menor, enquanto nas versões AGM, os tempos foram superiores ao *MagPIe-like*, devido à necessidade de troca de mensagens para a construção das suas árvores. A segunda coluna, mostra o atraso imposto, já citado acima, para a inclusão do novo valor de latência. Para os tempos de execução da operação MPI_Bcast, nota-se que as versões AGM-adap e AGM-adapII obtiveram tempos muito inferiores em relação aos tempos gastos pela AGM e *MagPIe-like*. As duas primeiras versões foram, aproximadamente, 21 vezes mais rápidas que a AGM e 85 vezes mais rápidas do que a *MagPIe-like*.

Durante a execução destes testes, os seguintes canais tiveram seus valores alterados: (04,06): 1,00 → 9999,00 e (12,20): 35,00 → 45,00 com aumento das latências; (00,02): 120001,00 → 6000 e (12, 16): 30000,00 → 21,00 tiveram os valores das latências reduzidos.

A quarta coluna apresenta os tempos gastos para a análise das latências e adaptação da árvore, lembrando que a versão AGM-adap executa a cada *bcast* a análise de alteração das latências, enquanto a versão AGM-adapII, nestes testes, fez esta análise a cada quatro *broadcasts* executados. Isto justifica o seu melhor desempenho nos tempos gastos para adaptação da árvore e, conseqüentemente, em toda a aplicação. A última coluna informa o tempo total de execução da aplicação, o que leva a concluir que, mesmo com a inclusão da análise das latências e da adaptação da árvore, as versões AGM-adap demonstram ter um desempenho melhor em relação ao *MagPIe-like*.

Figura 5. Atualização da árvore com variação no valor das latências superior a 10%.

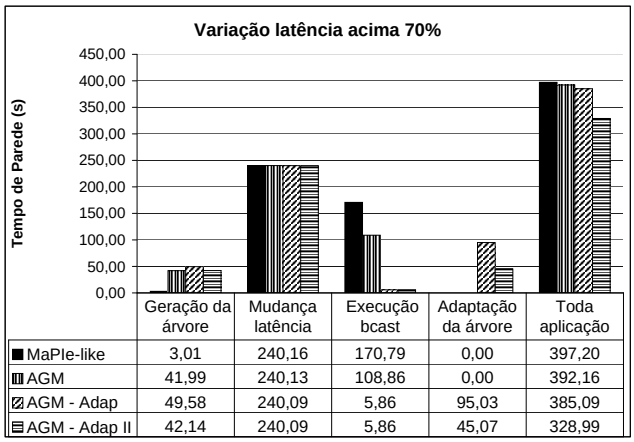


	Geração da árvore	Mudança latência	Execução bcast	Adaptação da árvore	Toda aplicação
MaPIe-like	3,01	240,16	170,79	0,00	397,20
AGM	41,99	240,13	108,86	0,00	392,16
AGM - Adap	41,93	240,12	5,86	111,89	393,83
AGM - Adap II	42,03	240,09	5,86	74,31	357,94

Na Figura 6, as mesmas observações feitas para o teste anterior se aplicam aqui, acrescentando apenas o fato da variação das latências ter de ser igual ou superior a 70% do valor original da latência, para que seja realizada a atualização da árvore. Desta forma, o canal (12,20) continua pertencendo à árvore, pois, o aumento do seu valor foi próximo a 28,5%. Como uma adaptação deixa de ser realizada, em relação ao teste anterior, observa-se uma melhoria nos tempos de adaptação da árvore e de toda a aplicação para as versões AGM-adap e AGM-adapII.

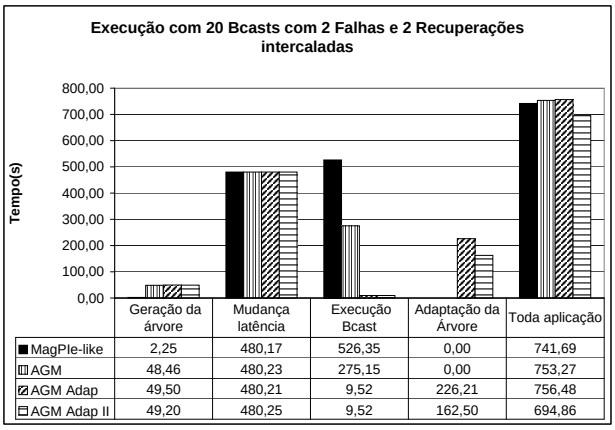
Outro experimento realizado foi a execução de 20 *broadcasts* consecutivos, Figura 7, onde falhas e recuperações ocorreram em momentos distintos. Aqui a porcentagem de variação foi igual a 0%, ou seja, qualquer variação, mesmo que pequena foi tratada como uma falha ou recuperação. A necessidade de atualização da árvore ocorreu durante a execução do 4º, 8º, 12º e 16º *broadcast*, quando foram identificadas, uma falha, uma recuperação, outra falha e outra recuperação de canal, respectivamente. Pode-se observar que a execução da operação MPI_Bcast continua mostrando um desempenho muito superior, para as versões AGM, quando comparadas com a *MagPIe-like*. O tempo total da aplicação para a versão AGM-adap ficou um pouco maior que o *MagPIe-like*, devido, principalmente, ao custo da manutenção da árvore. Em relação à versão AGM-adapII, o tempo da aplicação mostrou-se melhor com a diminuição do tempo na análise de latências e adaptação da árvore.

Figura 6. Atualização da árvore com variação no valor das latências superior a 70%.



	Geração da árvore	Mudança latência	Execução bcast	Adaptação da árvore	Toda aplicação
MaPIe-like	3,01	240,16	170,79	0,00	397,20
AGM	41,99	240,13	108,86	0,00	392,16
AGM - Adap	49,58	240,09	5,86	95,03	385,09
AGM - Adap II	42,14	240,09	5,86	45,07	328,99

Figura 7. Comparação dos algoritmos para execução com 20 bcast's



	Geração da árvore	Mudança latência	Execução Bcast	Adaptação da Árvore	Toda aplicação
MagPIe-like	2,25	480,17	526,35	0,00	741,69
AGM	48,46	480,23	275,15	0,00	753,27
AGM Adap	49,50	480,21	9,52	226,21	756,48
AGM Adap II	49,20	480,25	9,52	162,50	694,86

6. Conclusões

Foi observado um ganho de desempenho significativo para a disseminação da informação quando utilizamos a ferramenta aqui proposta, que considera as variações nos valores das latências nos canais. Nota-se, entretanto, que ainda há um custo elevado associado à atualização da estrutura de árvore, principalmente, em relação à análise dos dados coletados pelo NWS.

A implementação de outras operações coletivas para a ferramenta proposta, utilizando estruturas dinâmicas, e a investigação de técnicas para redução dos custos de análise das latências coletadas pelo NWS constituem tópicos de interesse em trabalhos futuros.

Referências

[1] C. Cheng, I. Cimet, and S. Kumar. A protocol to maintain a minimum spanning tree in a dynamic topology. *SIGCOMM Comput. Commun. Rev.*, 18(4):330–337, 1988.

[2] M. den Burger, T. Kielmann, and H. E. Bal. Topomon: A monitoring tool for grid network topology. In *ICCS '02: Proceedings of the International Conference on Computational Science-Part II*, pages 558–567, London, UK, 2002. Springer-Verlag.

[3] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[4] P. J. Husbands and J. C. Hoe. MPI-StarT: Delivering network performance to numerical applications. In *SC'98, Nov*, 1998.

[5] N. Karonis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *ArXiv Computer Science e-prints*, June 2002.

[6] N. T. Karonis, B. de Supinski, I. Foster, W. Gropp, and E. Lusk. A multilevel approach to topology-aware collective operations in computational grids. *ArXiv Computer Science e-prints*, June 2002.

[7] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MAGPIE: MPI's collective communication operations for clustered wide area systems. *ACM SIGPLAN Notices*, 34(8):131–140, Aug. 1999.

[8] T. L. M. O. S. Lab. Lam/mpi user's guide - version 7.1.1. 2004.

[9] S. Lacour. Mpich-g2 collective operations: Performance evaluation, optimizations. September 2001.

[10] I. F. Nicholas T. Karonis, Bronis R. De Supinski and W. Gropp. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. pages 377–384, 2000.

[11] N. T. S. Rich Wolski and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15:757–768, 1999.

[12] H. Saito, K. Taura, and T. Chikayama. Collective operations for wide-area message passing systems using adaptive spanning trees. *6th IEEE/ACM International Workshop on Grid Computing*, pages 40–48, 2005.

[13] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[14] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Perform. Eval. Rev.*, 30(4):41–49, 2003.

[15] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: the network weather service. In *Supercomputing '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–19, New York, NY, USA, 1997. ACM Press.