Usando Performance Relativa para Avaliar Desempenho em Supercomputadores Paralelos

Josilene Aires Moreira
Dataprev – Empresa de Tecnologia e Informações da Previdência Social
josi.aires@pb.previdenciasocial.gov.br

Walfredo Cirne
Universidade Federal de Campina Grande
walfredo@dsc.ufpb.br

Resumo

Em principio, a performance de um supercomputador paralelo é o resultado da atuação do escalonador sobre a workload que está sendo processada. Entretanto, dependendo da métrica utilizada para representar este desempenho, o resultado final da avaliação pode variar, gerando conflitos. Dessa forma, a métrica de performance é também um fator que influencia nos resultados do processo de avaliação. A métrica Performance Relativa, aqui apresentada, através da comparação dos turn-around times de todo o conjunto de programas obtidos pelos diferentes métodos de escalonamento, possibilita minimizar a influência que a métrica possa exercer sobre o resultado da avaliação de performance. Possibilita também identificar e estudar os subconjuntos de dados onde cada método obteve melhor desempenho, a fim de determinar suas características.

1. Introdução

A avaliação de performance de supercomputadores paralelos não é uma tarefa fácil, pois há uma multiplicidade de fatores envolvidos cujas interações interferem no processo de avaliação, afetando os resultados.

Em princípio a performance deveria depender apenas da *workload* (conjunto de programas) que está sendo processada e do método de escalonamento utilizado para gerenciar a fila de execução dos programas [1] [2] [3]. Mas há exemplos na literatura que mostram outro aspecto da avaliação influenciando nos resultados: a métrica de performance [2] [4] [5].

Acreditamos que boa parte do problema desta interferência seja devido à natureza sumarizadora das métricas baseadas em estatísticas utilizadas até então. A sumarização oferecida por uma estatística perde muita informação, uma vez que as distribuições dos *turn-around times* são tipicamente muito elásticas nas *workloads* dos supercomputadores paralelos [3]. Propomos como solução o uso da Performance Relativa, que baseia-se em toda a distribuição dos *turn-around times* (tempo passado entre a submissão e o término de um programa) obtidos pelos escalonadores avaliados.

A análise de performance baseada em Performance Relativa parece sofrer menos interferência da *workload* do que as métricas baseadas em estatísticas sumarizadoras, uma vez que seu resultado inclui a análise de toda a distribuição e não usa sumarização dos dados. Possibilita também uma boa visualização gráfica comparativa dos resultados.

Nosso estudo de caso investiga o controvertido problema *Easy Backfilling X Conservative Backfilling* [6] [7]. Os artigos [6] e [7] analisam a performance de computadores IBM SP2 utilizando ambas as formas de *Backfilling*: *Easy* e *Conservative*. No artigo [6] concluise que a performance dos dois algoritmos é similar, sendo que *Conservative* possui a vantagem da previsibilidade. Em [7], chega-se à conclusão que, de forma geral, as *workloads* provenientes dos computadores SP2 favorecem o *Easy Backfilling*. Porém em algumas métricas o resultado parece se inverter. Fica no ar, portanto, a pergunta: Qual métrica vale de fato?

Estes resultados controvertidos e inconclusivos hoje conhecidos nos motivaram a utilizar esta nova maneira de comparar desempenho, Performance Relativa, a fim de avaliarmos o mesmo problema.

As *workloads* que usamos são *traces* de IBM SP2 dos centros de supercomputação KTH, SDSC e CTC, como também uma *workload* sintética. Como veremos em detalhes, nossa análise mostra que *Easy Backfilling* de fato proporciona uma melhor performance que *Conservative Backfilling*. Mais ainda, conseguimos identificar que *Easy* favorece programas grandes, explicando assim as razões pelas quais *Easy* obtém melhor performance nos cenários avaliados.

Este artigo está organizado da seguinte forma. A seção 2 descreve os dois métodos de escalonamento utilizados no nosso estudo de caso. A seção 3 comenta as métricas de performance existentes e apresenta a Performance Relativa. A seção 4 apresenta o estudo de caso realizado e os resultados alcançados. Finalmente a seção 5 reúne as conclusões obtidas.

2. Escalonadores

Em um ambiente de processamento paralelo um programa deve explicitar o número de processadores (*nodes*) e o tempo requerido (*tr*) para a sua execução [8]. Note que o programa é abortado pelo sistema caso ele exceda o tempo requerido. O produto entre o número de processadores e o tempo requerido é conhecida como *forma* ou *área* do programa. Um programa submetido pode não encontrar recursos disponíveis para ser executado imediatamente. Quando isso acontece, o programa é colocado em uma fila, a qual é controlada pelo escalonador do supercomputador. O escalonador tem a função de receber as requisições dos programas e decidir quando estes iniciarão a sua execução e que processadores usarão. Obviamente, dependendo da política de escalonamento escolhida, os sistemas podem apresentar variações de desempenho [8].

Backfilling é uma otimização do método FCFS (*First Come First Served*), onde os programas que chegam para executar são colocados em uma fila e alocados por ordem de chegada. Como os programas requerem um determinado número de processadores para executar (*nodes*), pode acontecer de haver processadores disponíveis, mas não em quantidade suficiente para que o primeiro programa na fila possa executar. *Backfilling* tenta reduzir ao mínimo esta subutilização de recursos através da alocação de outros programas da fila capazes de preenchê-los [7].

2.1. *Conservative Backfilling*

No algoritmo do *Conservative Backfilling*, quando o escalonador encontra o primeiro programa que não dispõe de recursos para ser executado imediatamente, percorre a fila procurando um outro que possa ser executado com os recursos que estão disponíveis no momento. Este programa pode então ser adiantado, desde que não atrase o início estimado de nenhum outro ocupante da fila de espera.

Deste modo, o algoritmo garante previsibilidade, pois no momento da submissão já é possível garantir um limite máximo para o início de um programa (máximo porque os programas à sua frente na fila poderão terminar antes ou então o próprio programa poderá ser adiantado através de *backfilling*).

2.2. *Easy Backfilling*

O algoritmo de *Easy Backfilling* também percorre a fila no momento em que um programa não pode ser executado, a fim de encaixar um outro que aproveite os recursos disponíveis. A diferença é que este método é mais agressivo, pois permite adiantar qualquer programa da fila de espera desde que isto não cause atraso ao primeiro da fila [7].

Este algoritmo ganha em agressividade, propondo uma maior utilização do sistema, mas perde em previsibilidade, pois é impossível estabelecer qualquer garantia de quando um programa será executado.

3. Métricas de performance

As métricas são os métodos utilizados para avaliar o desempenho de um sistema. Existem algumas métricas comumente usadas para representar a performance dos supercomputadores paralelos, cada uma com as suas características.

3.1. *Turn-aroud time* médio

O *turn-around time* (*tt*), também conhecido como tempo de resposta, é definido como o intervalo entre o instante em que o programa é submetido ao sistema até o momento em que toda a sua saída é produzida e o programa termina.

Para representar a performance do sistema quanto à execução de um conjunto de programas têm-se utilizado a média dos *turn-around times*, com a intenção de sumarizar estes valores através de um único número que represente performance. Porém a média possui a característica de ser influenciada por valores extremos [9]. Uma *workload* que possua a maioria dos programas com *turn-around time* curto e alguns poucos programas com *turn-around time* muito longo, certamente terá sua média dos *turn-around times* mais influenciada por estes valores muito longos, o que elevará a média geral. Por exemplo, o *turn-around time* médio para noventa e nove programas de 1 segundo e um programa de 2000 segundos é 20.99 segundos.

3.2. *Slowdown* médio

O *slowdown*, também conhecido como fator de expansão, é o *turn-around time* (*tt*) normalizado pelo tempo de execução (*te*), $s = tt / te$ [10]. O *slowdown* é utilizado a fim de reduzir valores extremos associados a programas muito longos. A idéia principal por trás do *slowdown* é que um programa espere na fila um tempo proporcional ao seu tempo de execução. Por outro lado, o *slowdown* supervaloriza a importância dos programas muito curtos. Por exemplo, um programa de um segundo que espera dez minutos na fila tem um *slowdown* de 600.

3.3. *Bounded slowdown* médio

O *bounded slowdown* procura atenuar o *slowdown* através da definição de um limite para o tempo de execução $bs = tt / max(te, \tau)$ [10]. A diferença é que, para programas muito pequenos, o *slowdown* será limitado pelo valor limite (τ) escolhido, e não pelo tempo de execução. Obviamente, o comportamento desta métrica vai depender do valor de τ. Um valor tipicamente utilizado é $\tau =$

10 [1] [10]. Por exemplo, para o mesmo programa de um segundo que espera dez minutos na fila, o *bounded slowdown* é de 60.

A utilização tanto do *slowdown* como do *bounded slowdown* levam a novos problemas, pois podem encorajar o escalonador a fazer com que os programas tenham um tempo de execução maior a fim de baixar os valores destas métricas.

3.4. Média geométrica do *turn-around time*

A média geométrica é definida por $medgeom(x_1,....,x_n) = \sqrt[n]{x_1 * ... * x_n}$, e tem a característica de ser menos influenciada por valores extremos [8]. Desta forma, a idéia de usar média geométrica ao invés da média aritmética tem o objetivo de reduzir o efeito dos programas com *turn-around time* muito longo ou muito curto sobre a média geral dos *turn-around times* da *workload*. Esta métrica é usada para agregar o tempo de execução dos programas que compõem o *Spec Benchmark* [11].

3.5. Performance relativa

A performance relativa toma como base a relação entre os dois valores de *turn-around time* obtidos para cada programa através das duas alternativas de escalonamento que se deseja comparar. A metodologia utilizada é a seguinte:

- Define-se quais escalonadores (ou supercomputadores) deseja-se comparar, dois a dois (chamemos de **a** e **b**);
- Toma-se uma determinada *workload* de tamanho **n** a ser submetida aos dois escalonadores;
- Registra-se o *turn-around time* de cada programa obtido em cada escalonador (tt_a e tt_b);
- Calcula-se a razão entre os **n** pares dos *turn-around times*, (tt_a / tt_b);
- Finalmente, calcula-se a distribuição acumulada de freqüência das razões obtidas, a fim de obter a representação gráfica.

Vejamos o resultado na figura 1, através da distribuição acumulada de freqüência.

Para cada razão **i** interpreta-se o resultado da seguinte maneira:

- Se tt_a / tt_b = **1**, o programa obteve o mesmo *turn-around time* nos dois escalonadores;
- Se tt_a / tt_b < **1** o programa obteve melhor *turn-around time* no escalonador **a**;
- Se tt_a / tt_b > **1** o programa obteve melhor *turn-around time* no escalonador **b**.

Através do gráfico percebemos que aproximadamente 65% dos programas obtiveram o mesmo valor de *turn-around time* nos dois escalonadores; o escalonador **a** foi melhor em aproximadamente 11% dos *turn-around times* e o escalonador **b** foi melhor em aproximadamente 24% dos *turn-around times*. Mais ainda, a partir do gráfico é possível avaliar o impacto que os escalonadores **a** e **b** tiveram sobre os programas que beneficiaram. Por exemplo, alguns programas chegaram a concluir em tempo 32.000 vezes menor quando usando o escalonador **a**. Por outro lado, o escalonador **b** concluiu alguns programas em tempo 1.000.000 de vezes menor que o escalonador **a**.

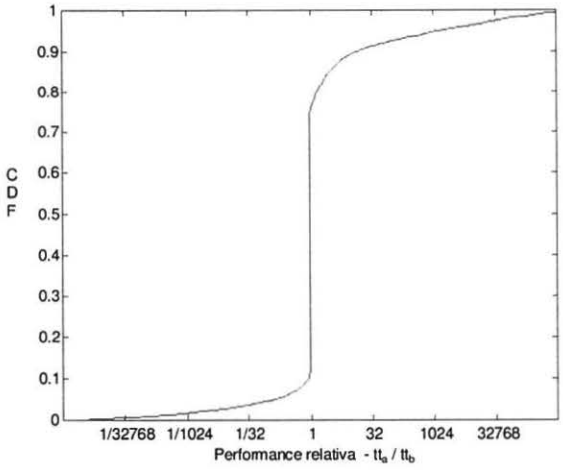

Figura 1 - Exemplo de performance relativa

O uso desta métrica é interessante porque analisa toda a distribuição da *workload*, ao contrário das métricas baseadas em estatísticas. Assim, as características da *workload*, tais como ser dominada por programas muito longos ou ainda possuir valores extremos que afetem a média aritmética dos *turn-around times* não interferem no resultado desta métrica, uma vez que a comparação é feita programa a programa.

Este mecanismo de avaliação permite que se conheça exatamente em quanto um algoritmo de escalonamento foi mais eficiente do que outro para uma *workload*, além de permitir identificar para quais programas. A partir destas informações é possível isolar o conjunto de programas onde cada algoritmo foi melhor, a fim de identificar e estudar suas características, como veremos adiante no estudo de caso.

Como desvantagem, podemos citar que para comparar mais de dois escalonadores é necessário fazê-lo dois a dois, obtendo sempre a relação de um *turn-around time* para com outro.

4. Estudo de caso

Realizamos a simulação dos métodos de escalonamento *Conservative Backfilling* e *Easy Backfilling* a fim de efetuar o processo de avaliação de performance destes algoritmos através de algumas métricas conhecidas e

também da Performance Relativa. Foram usados três *logs* de *workloads* provenientes de diferentes centros de supercomputação disponíveis na *web* [12] [13] e ainda uma quarta *workload* sintética obtida através de um modelo de carga [14].

4.1. O ambiente da simulação

O nosso simulador recebe como entrada uma *workload* contendo para cada programa a quantidade de processadores (*nodes*), o *request time* (*tr*), o momento da submissão e o tempo de execução (*te*). Também são fornecidos ao simulador o método de escalonamento e a descrição do supercomputador, incluindo o número de processadores disponíveis.

O processamento destes programas é simulado através dos dois métodos de escalonamento descritos na seção 2, obtendo-se os *turn-around times* como resultado. Estes *turn-around times* são então comparados através das métricas:

- *Turn-around time* médio (*tt* médio);
- *Slowdown* médio;
- *Bounded slowdown* médio com $\tau = 10$;
- *Bounded slowdown* médio com $\tau = 100$;
- Média geométrica e
- Performance relativa.

4.2. *Workloads* utilizadas

Com o objetivo de tornar a simulação o mais realista possível, foram utilizados *traces* de supercomputadores de diferentes fontes disponíveis na *web* [12][13], além de um modelo sintético [14].

Tabela 1 – Descrição das workloads

Nome	máquina	processadores	Programas	período
CTC	*Cornell Theory Center SP2*	430	79.296	Jul 1996 Mai 1997
KTH	*Swedish Royal Institute of Technology SP2*	100	28.474	Set 1996 Ago 1997
SDSC	*San Diego Supercomputer Center SP2*	128	16.376	Jan 1999 Mai 1999
SYNT	Modelo Sintético	800	50.000	-

Nem todos os programas que são submetidos aos supercomputadores completam a sua execução. Alguns são cancelados pelo usuário. Como estamos analisando as métricas baseadas nos *turn-around times* dos programas, retiramos aqueles cancelados das *workloads* CTC, SDSC e do modelo sintético a fim de considerarmos só os programas que completaram sua execução ou foram interrompidos pelo escalonador por exceder o tempo requisitado (*tr*). A *workload* KTH não faz discriminação entre os programas cancelados e os completados.

4.3. Resultados da simulação

As tabelas 2 a 6 mostram os resultados de performance obtidos para cada método de escalonamento avaliado segundo as diversas métricas.

Tabela 2 – *Turn-around time* médio

tt médio		
Dados	EASY	CONS
CTC	11755	13409
KTH	14893	16202
SDSC	20158	20008
SYNT	10321	10695

Tabela 3 – Média geométrica do *tt*

Média geométrica do *tt*		
Dados	EASY	CONS
CTC	1846	2323
KTH	1628	2133
SDSC	1474	1738
SYNT	1483	1561

Tabela 4 – *Slowdown* médio

Slowdown médio		
Dados	EASY	CONS
CTC	5.32	13.97
KTH	158.11	209.35
SDSC	59.50	61.46
SYNT	67.99	42.52

Tabela 5 – *Bounded slowdown* médio (τ=10)

Bounded slowdown médio		
Dados	EASY	CONS
CTC	5.32	13.97
KTH	79.44	92.26
SDSC	59.50	61.46
SYNT	25.36	16.47

Tabela 6 – Bounded slowdown médio (τ=100)

Bounded slowdown médio		
Dados	EASY	CONS
CTC	4.36	8.60
KTH	20.55	20.82
SDSC	43.60	38.28
SYNT	7.65	5.64

Os valores das tabelas 2 a 6 estão expressos em segundos, com as colunas EASY mostrando o valor obtido para o método *Easy Backfilling* e a coluna CONS trazendo o valor obtido para o *Conservative Backfilling*.

A tabela 5 mostra o resultado do *bounded slowdown* calculado com $\tau = 10$; como tanto para CTC como para SDSC não haviam programas com *turn-around time* menores que 10, foi calculado novamente o *bounded slowdown* com $\tau = 100$, cujos valores estão na tabela 6.

Analisando os resultados destas tabelas comparativamente temos:

- Através da métrica média geométrica, *Easy* obteve melhor performance em todas as *workloads*.
- Através da métrica *turn-around time* médio, o método *Conservative* foi ligeiramente melhor para a *workload* SDSC enquanto *Easy* foi melhor para as outras *workloads*.
- Através da métrica *slowdown* médio, o método *Conservative* foi melhor para a *workload* sintéti-

ca, enquanto *Easy* foi melhor para as outras *workloads*.

- Através do *bounded slowdown médio* com $\tau = 10$, *Conservative* foi melhor para a *workload* sintética, enquanto *Easy* foi melhor para as outras *workloads*.
- Através do *bounded slowdown médio* com $\tau = 100$, *Conservative* foi melhor para a *workload* sintética e para SDSC, enquanto *Easy* foi melhor para CTC e ligeiramente melhor para KTH.

Percebemos que para estas *workloads* os resultados de performance apresentaram resultados conflitantes dependendo da métrica utilizada, o que já era esperado, pois há outros exemplos na literatura identificando e tentando esclarecer estes conflitos [4] [6] [7].

Com o objetivo de contribuir para esclarecer melhor estes resultados contraditórios, utilizamos então a métrica performance relativa. O cálculo foi feito segundo a razão $tt_{conservative}$ / tt_{easy} obtida a partir dos *turn-around times* dos programas submetidos aos métodos de escalonamento *Conervative Backfilling* ($tt_{conservative}$) e *Easy* Backfilling (tt_{easy}). Os gráficos das figuras 2, 3, 4 e 5 foram traçados a partir da distribuição acumulada de freqüência destes valores.

A porção da curva à esquerda de 1 representa onde o método *Conservative Backfilling* foi melhor; a porção da curva à direita de 1 indica onde o método *Easy Backfilling* foi melhor. Os valores 1 indicam onde os dois métodos obtiveram o mesmo *turn-around time*.

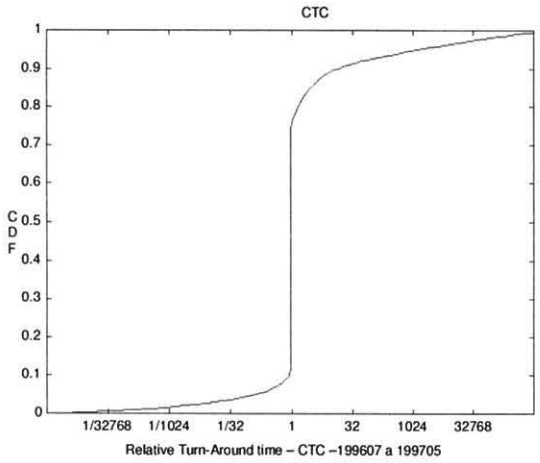

Figura 2 – Performance relativa CTC

- Na *workload* CTC vemos que o método *Conservative* foi melhor em aproximadamente 12% dos *turn-around times*, enquanto *Easy* foi melhor em aproximadamente 25%;
- Na *workload* KTH percebemos que o método *Conservative* foi melhor em aproximadamente 18% dos *turn-around times*, enquanto *Easy* foi melhor em aproximadamente 35%;
- Na *workload* SDSC temos que o método *Conservative* foi melhor em aproximadamente 23% dos *turn-around times*, enquanto *Easy* foi melhor em aproximadamente 33%;
- Na *workload* Sintética (SYNT) o método *Conservative* foi melhor em aproximadamente 18% dos *turn-around times*, *Easy* foi melhor em 35%.

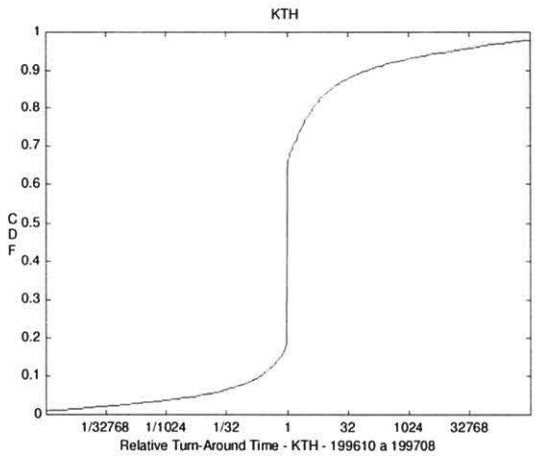

Figura 3 - Performance relativa KTH

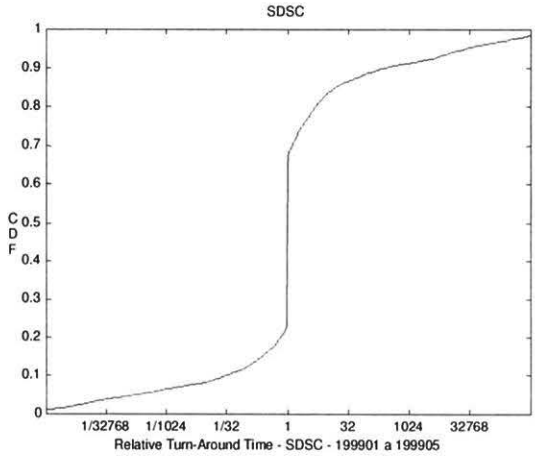

Figura 4 – Performance relativa SDSC

Outra análise importante que pode ser feita a partir dos gráficos é o quanto um algoritmo foi melhor em relação ao outro para os programas que beneficiou. Por exemplo, na *workload* CTC alguns programas chegaram a concluir em tempo 32.000 vezes menor quando usando o escalonador *Conservative*. Por outro lado, o escalonador *Easy* concluiu alguns programas em tempo 1.000.000 de vezes menor que o escalonador *Conservative*.

Um outro resultado surpreendente foi a quantidade de programas com o mesmo *turn-around time* nos dois métodos, variando de 44% em SDSC até 63% em CTC.

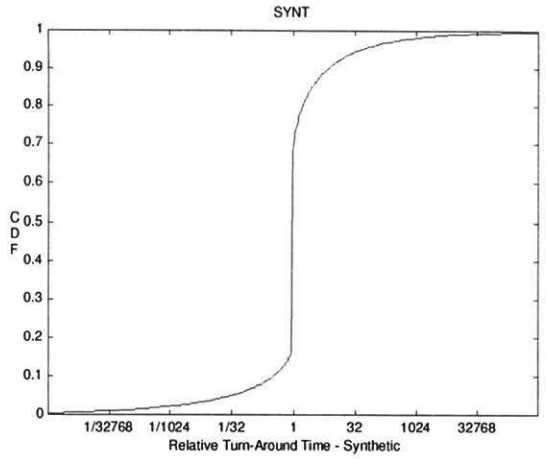

Figura 5 - Performance relativa SYNT

A segunda etapa dos resultados compreendeu o estudo do comportamento do conjunto de programas, a fim de identificar quais características eram favorecidas por um determinado método de escalonamento. As características estudadas foram as seguintes:

- número de processadores (*nodes*);
- tempo requisitado (*tr*);
- área requisitada (*tr X nodes*);
- tempo de execução (*te*) e
- área de execução (*te X nodes*).

Em cada *workload,* os programas foram ordenados por uma determinada característica e então divididos em três subconjuntos de igual tamanho: menores, médios e maiores. Para cada subconjunto traçamos então a Performance Relativa baseada nos *turn-around times* destes programas. Consideramos o resultado obtido pelo estudo da área (*tr X nodes*) como o mais interessante, pois é a partir dela que o escalonador trabalha para alocar os programas.

Encontramos uma relação entre o aumento desta área e a eficiência do *Easy Backfilling*: o desempenho do método melhorou à medida que a área aumentou, para todas as *workloads* consideradas. O resultado pode ser visualizado nas tabelas 7, 8, 9, 10 e nas figuras 6, 7, 8 e 9.

Os resultados em cada tabela estão expressos em valores percentuais: na tabela 7 (CTC) temos que para os programas pequenos *Easy* foi melhor em 13.63% dos casos e *Conservative* foi melhor em 13.51%, o que representa praticamente o mesmo desempenho. À medida que a área dos programas cresceu, o percentual onde *Easy* foi melhor aumentou, chegando a obter 35.52% contra 12.94% nos programas maiores.

A partir do gráfico da figura 6 esta situação pode ser visualizada: à medida que a área dos programas aumenta, o percentual onde *Easy* é melhor também aumenta. A variação do *Conservative* foi pequena para estes subconjuntos.

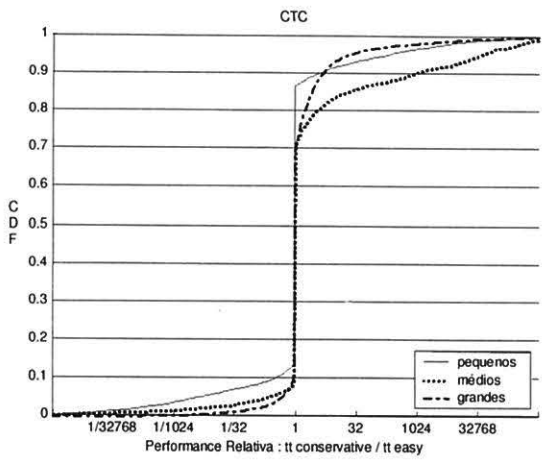

Figura 6 – Performance Relativa por área CTC

Tabela 7 - Performance relativa por área CTC

Performance relativa por área CTC			
Dados	EASY	CONS	IGUAL
Pequenos	13.63	13.51	72.84
Médios	30.84	9.59	59.56
Grandes	32.52	12.94	54.53

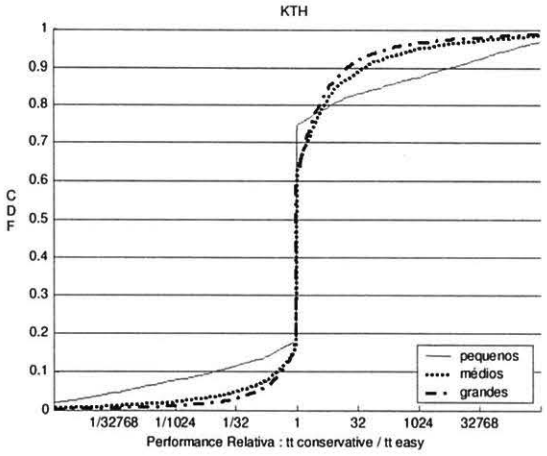

Figura 7 – Performance relativa por área KTH

Tabela 8 - Performance relativa por área KTH

Performance relativa por área KTH			
Dados	EASY	CONS	IGUAL
Pequenos	25.92	18.51	55.55
Médios	38.39	17.19	44.41
Grandes	41.34	20.05	38.60

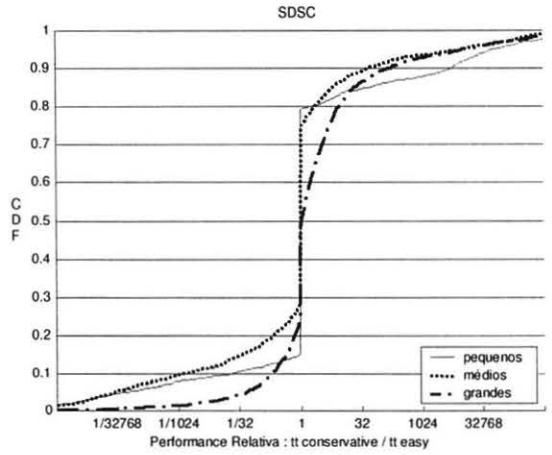

Figura 8 – Performance Relativa por área SDSC

Tabela 9 - Performance relativa por área SDSC

Performance relativa por área SDSC			
Dados	EASY	CONS	IGUAL
Pequenos	20.95	15.27	63.77
Médios	26.53	29.19	44.27
Grandes	52.00	26.38	21.61

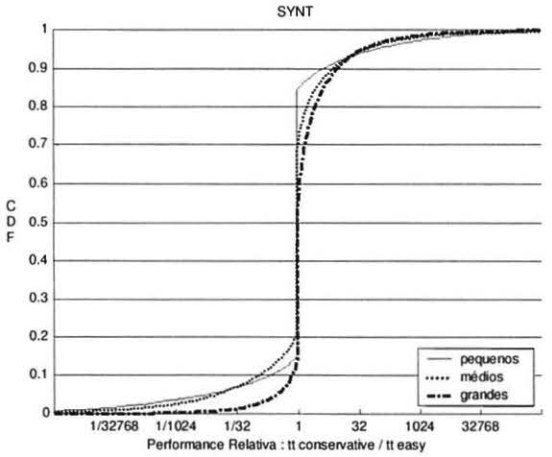

Figura 9 – Performance relativa por área SYNT

Tabela 10 - Performance por área SYNT

Performance Relativa SYNT			
Dados	EASY	CONS	IGUAL
Pequenos	15.68	15.48	68.82
Médios	31.48	22.97	45.53
Grandes	48.34	15.84	35.81

Em todas as *workloads* utilizadas observamos um comportamento semelhante: o método *Easy* foi melhorando o desempenho, isto é, obtendo um percentual crescente de melhores *turn-around times* à medida que a área dos programas aumentou. Esta constatação justifica-se devido à maior possibilidade dos programas com maior área serem adiantados na fila através do algoritmo *Easy* do que pelo *Conservative*, pois neste último um programa só poderá passar à frente dos outros se não atrasar o tempo de início estimado de nenhum programa da fila.

Com relação aos *turn-around times* obtidos pelo método *Conservative* não se pôde determinar nenhum crescimento ou decréscimo em função da área dos programas para as *workloads* estudadas.

Vimos então através do estudo de caso que o método de escalonamento *Easy backfilling* realmente proporcionou uma melhor performance para as *workloads* analisadas. Também vimos que *Easy* favorece programas grandes, o que pôde ser observado através da análise das características dos programas favorecidos por cada método de escalonamento.

5. Conclusões

A obtenção de um resultado correto em uma avaliação de performance em supercomputadores paralelos não é uma tarefa fácil. Além da interação entre os métodos de escalonamento e as *workloads*, temos também a interferência da métrica.

A métrica escolhida para representar o desempenho do sistema pode influenciar fortemente no processo de avaliação, pois as interações com a *workload* geram resultados que podem estar refletindo as características da carga e modificando as conclusões da avaliação. Um exemplo disto são as métricas baseadas em estatísticas sumarizadoras, que podem ter seus resultados afetados por valores extremos da *workload*.

A Performance Relativa, através da comparação direta dos *turn-around times* dos métodos de escalonamento, pretende minimizar esta interferência através de uma representação dos resultados que não seja afetada pelos valores extremos. Esta propriedade de não ser afetada é conseguida através da comparação dos *turn-around times* programa a programa, abrangendo toda a distribuição da *workload*. A visualização gráfica proporcionada pela métrica também possibilita analisar o impacto que os escalonadores tiveram sobre os programas que beneficiaram.

Como estudo de caso usamos *Easy Backfilling* x *Conservative Backfilling*, um caso em que a literatura apresenta resultados conflitantes [6] [7], e pudemos concluir claramente que *Easy Backfilling* proporcionou melhor desempenho para a maioria dos programas das *workloads* analisadas. Descobrimos também que *Easy* beneficia os programas grandes.

Mas talvez a contribuição mais importante deste trabalho seja metodológica, apresentando uma métrica que,

por não usar estatísticas sumarizadoras, é fundamentalmente diferente das métricas até então utilizadas.

Obviamente, o uso desta nova métrica não invalida aquelas anteriormente utilizadas, pois o processo de avaliação de performance não tem como objetivo único concluir quem é melhor. Mais do que isso, estes estudos têm sentido à medida que nos ajudam a compreender cada vez mais o comportamento dos escalonadores ao lidar com determinadas características das *workloads*. Neste sentido, seria indicado conduzir qualquer avaliação de performance usando o maior número de métricas e os mais completos modelos de carga disponíveis, e não apenas um em particular. Se os resultados obtidos forem conflitantes, deve-se analisar detalhadamente o porquê e procurar conhecer as causas dos conflitos, isolando-os e refazendo o processo com os dados e métricas que forem mais significativos para o objetivo do estudo.

7. Referências

[1] D. Feitelson and L. Rudolph, ***Metrics and Benchmarking for Parallel Job Scheduling***. In Job Scheduling Strategies for Parallel Processing, Dror Feitelson and Larry Rudolph (Eds.), Springer-Verlag, Lecture Notes in Computer Science vol. 1459, pp. 1-24, 1998.

[2] D. Feitelson, ***The Effect of Metrics and Workloads on Performance Evaluation.*** Technical Report 2001-13, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Oct 2001.

[3] V. Lo, J. Mache, and K. Windisch. ***A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling***. In Job Scheduling Strategies for Parallel Processing, Dror Feitelson and Larry Rudolph (Eds.), Springer Verlag, Lect. Notes Comput. Sci. vol. 1459, pp. 25-46,1998.

[4] D. Feitelson. ***Analyzing the Root Causes of Performance Evaluation Results.*** Technical Report 2002-4, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Mar 2002.

[5] S. Chapin, W. Cirne, D. Feitelson, J. Jones, S. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. ***Benchmarks and Standards for the Evaluation of Parallel Job Schedulers.*** In Job Scheduling Strategies for Parallel Processing, D. Feitelson and Larry Rudolph (Eds.) Springer-Verlag, Lecture Notes in Computer Science, vol. 1659, pp. 66-89, 1999.

[6] D. Feitelson and A. Mu'Alem. ***Utilization and Predictability in Schedulling IBM SP2 with Backfilling.*** In 12th Intl. Parallel Processing Symp., pp 542-546, Apr 1998.

[7] A. Mu'Alem and D. Feitelson. ***Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling.*** IEEE Trans. Parallel & Distributed Syst, 12(6), pp. 529-543, Jun 2001.

[8] W. Cirne, ***Using Moldability to Improve the Performance of Supercomputer Jobs.*** Ph.D. Thesis, University of California, San Diego. 2001.

[9] P. L. Costa Neto. ***Estatística***. Editora Edgard Blücher Ltda,1977.

[10] D. Feitelson, ***Metrics for Parallel Job Scheduling and Their Convergence***. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, Lecture Notes in Computer Science Vol. 2221, pp. 188-206, 2001.

[11] ***Standard Performance Evaluation Corporation***. The SPEC web page. http://www.spec.org/

[12] Dror Feitelson. ***The Parallel Workloads Archive***. http://www.cs.huji.ac.il/labs/parallel/workload/

[13] Victor Hazlewood. ***NPACI JobLog Repository***. http://joblog.npaci.edu/

[14] W. Cirne and F. Berman. ***A Comprehensive Model of the Supercomputer Workload***. In Proceedings of WWC-4: IEEE 4th Annual Workshop on Workload Characterization, Dec 2001.