# EASAHE, Um algoritmo para o agendamento de trabalhos em ferramentas de processamento de dados com preocupações de eficiência energética

Renato Azevedo[1] and Ricardo Vilaça[1][0000−0002−6957−1536]

HASLab - High-Assurance Software Lab, INESC TEC & U. Minho, Portugal
http://www.inesctec.pt
renato.a.azevedo@inesctec.pt, rmvilaca@di.uminho.pt

**Resumo** As ferramentas de processamento de dados massivos em ambientes distribuídos como o Spark ou Dask permitem aos programadores efectuarem processamento sobre quantidades massivas de dados em grandes clusters.
As ferramentas atuais utilizam algoritmos simples para o agendamento eficiente de trabalhos de processamento de dados em computação distribuída, recorrendo a heurísticas sem ter em conta as características da carga de trabalho. Trabalho recente explora o agendamento eficiente de trabalhos de processamento de dados em computação distribuída.
Neste artigo propomos um novo algoritmo para o agendamento de trabalhos para ferramentas de processamento de dados massivos com preocupações de eficiência energética. A implementação num simulador e avaliação usando *traces* de execuções reais e sintéticas em Spark, demonstram que o algoritmo consegue reduzir o consumo energético em até 11.5%, além de conseguir reduzir o tempo de execução dos trabalhos em até 11.9%, sem grande impacto no tempo gasto no agendamento.

**Keywords:** Spark · Agendamento · Eficiência Energética.

## 1 Introdução

Na última década, a popularização da internet levou a um aumento exponencial da quantidade de informação geradas. Para processar esta grande quantidade de dados, existe uma necessidade cada vez maior de construir infraestruturas com enorme poder de computação. No entanto, o desenvolvimento do poder de computação está a levar a um aumento cada vez maior do consumo energético, já que por um lado o hardware consume mais energia de forma a processar mais dados, e também porque estas infraestruturas necessitam de ser constantemente arrefecidas devido à grande quantidade de calor que libertam.

Ferramentas de processamento de dados como o Apache Spark [22] são bastante usadas neste contexto para processar dados em grande escala. No entanto, estas ferramentas não possuem qualquer tipo de opção para poupança de energia [14].

Sendo assim, uma boa forma para travar o aumento do consumo energético dos centros de dados é otimizando as ferramentas de processamento de dados, já que estas são responsáveis por uma grande parte do trabalho realizado nestas infraestruturas. Em particular, o desenvolvimento de um algoritmo de agendamento com preocupações de eficiência energética capaz de utilizar os recursos de forma mais eficiente, de um ponto de vista energético, será capaz de reduzir o consumo energético nos centros de dados.

Desta forma, neste artigo propomos o EASAHE, um novo algoritmo de agendamento de trabalhos para ferramentas de processamento de dados escaláveis em ambientes homogéneos, capaz de reduzir o consumo energético sem grandes impactos no tempo de execução. O objetivo do algoritmo é diminuir o consumo energético através da redução do número de recursos que não estão a ser utilizados ao máximo, o que permite não só melhorar a taxa de utilização dos recursos, mas também utilizar menos recursos do que normalmente seria necessário.

Este artigo está organizado da seguinte forma: a secção 2 mostra o estado da arte, a secção 3 apresenta e explica o algoritmo, a secção 4 explica o ambiente de simulação utilizado e em particular, como é feita a medição energética no mesmo, a secção 5 mostra os resultados obtidos, e por fim, a secção 6 faz uma conclusão e apresenta o trabalho futuro.

## 2 Trabalho relacionado

Uma vez que o problema de agendamento é NP-completo, como mostrado em [20], torna-se necessário desenvolver heurísticas para conseguir resolver este problema num tempo aceitável. Assim, nos últimos anos surgiram vários trabalhos sobre algoritmos de agendamento em diversos contextos, como resumido em [8]. Uma clara divisão entre estes algoritmos é relativamente a serem desenvolvidos para ambientes homogéneos ou heterogéneos, sendo a maioria desenvolvidos para funcionar em ambientes heterogéneos, de forma a tirar o máximo partido das diferenças entre as máquinas.

Alguns destes algoritmos propostos não têm em consideração o consumo energético, como o FIFO [23] e o FAIR [7] que estão implementados por defeito na ferramenta Apache Spark, além de outros como o HEFT [19] que foi pensado para ser aplicado em ambientes heterogéneos. No entanto, existem versões orientadas à eficiência energética destes algoritmos, como por exemplo o E-FIFO [12].

Já outros algoritmos foram desenvolvidos a considerar a eficiência energética. Entre estes, o EASAS [9], proposto por H. Li et al. , é um algoritmo para a ferramenta Apache Spark, pensado para funcionar em ambientes heterogéneos, e capaz de reduzir o consumo energético em cerca de 25 a 40%, sem violar restrições de tempo. Neste algoritmo, os *executors* são ordenados por um critério de avaliação, que representa a prioridade de colocar uma tarefa nesse *executor*, e tentam colocar o máximo de tarefas possíveis nesse *executor*.

W. Shi et al. [16] propôs o algoritmo TPCBFD e a sua versão orientada à eficiência energética EATPCBFD, ambos para ambientes heterogéneos. A versão EATPCBFD deste trabalho foi capaz de melhorar a eficiência energética média

na ferramenta Apache Spark em cerca de 77%, além de reduzir as violações das restrições de tempo. A estratégia utilizada nestes dois algoritmos envolve agrupar as tarefas pelas suas características, e os nodos pela sua performance, e por fim alocar cada tarefa ao nodo mais adequado.

Todos estes algoritmos são aplicados a ambientes heterogéneos, mas tipicamente os *clusters* de processamento de dados possuem máquinas homogéneas. Uma vez que estes algoritmos tentam tirar proveito das diversas propriedades dos *clusters* heterogéneos, não terão o mesmo nível de performance num *cluster* homogéneo onde as diversas máquinas não apresentam propriedades diferentes.

Além de alterar a ordem das tarefas e mudar a escolha da máquina em que a tarefa vai ser executada, é ainda possível reduzir o consumo energético através do ajuste da frequência dos processadores. Um dos trabalhos capaz de fazer isto foi o FAESS-DVF proposto por H. Li et al. [10]. Este algoritmo é capaz de alcançar uma poupança energética de até 29.5% comparado ao algoritmo por defeito do Spark em YARN (FIFO), além de satisfazer as restrições de tempo.

Wang et al. [21] propôs em 2010 o algoritmo PATC capaz de atingir uma poupança energética de 39.7% em testes simulados, ao agrupar tarefas de acordo com os seus custos de comunicação, e diminuir a frequência para tarefas não críticas.

Em 2014, Tang et al. propuseram o algoritmo DEWTS [18], que é capaz de reduzir o consumo energético até 46.5%. Neste algoritmo, o ajuste de frequência é utilizado para estender a duração das tarefas, de forma a manter o processador ocupado.

Outros algoritmos utilizam modelos de aprendizagem automática, e mais recentemente aprendizagem reforçada. Um algoritmo que utiliza aprendizagem automática foi proposto por Berral et al. [2], onde utilizam um modelo para prever se a realocação de um trabalho para um nodo diferente levaria a uma poupança energética. Assim, os autores foram capazes de reduzir o consumo energético ao mesmo tempo que respeitavam as restrições de tempo.

Quanto a aprendizagem profunda, Mao et al. [11] propuseram o Decima, uma ferramenta capaz de aprender algoritmos de agendamento complexos dado um determinado objetivo. Este trabalho utiliza um simulador para treinar o modelo, e é capaz de aprender com sucesso novos algoritmos apenas ajustando a recompensa.

Shi et al. [17] propuseram um algoritmo de agendamento de trabalhos para Spark em ambientes de computação em nuvem híbridos, com o objetivo de reduzir a utilização destes serviços. Este algoritmo foi capaz de reduzir a utilização do *cluster* em até 13.8%, além de melhorar a performance em cerca de 5.55%. O agente proposto neste trabalho é capaz de aprender as características dos trabalhos e do ambiente em que estes serão executados.

Por fim, alguns algoritmos utilizam o conceito de *Application Signature*, como em [15], onde se utiliza uma versão reduzida da aplicação para obter informações que podem ser utilizadas no processo de agendamento. Estas informações recolhidas foram depois utilizadas no processo de agendamento para reduzir o *makespan* aumentar a poupança energética da aplicação.

## 3 Algoritmo

A estratégia utilizada pelo EASAHE para reduzir o consumo energético consiste em minimizar o número de recursos que não estão a ser utilizados. Isto é alcançado através da priorização de tarefas que libertam dependências, de forma a ter sempre tarefas para ocupar os recursos, e também pela agregação de tarefas num conjunto de máquinas de forma a utilizar os recursos eficazmente. Esta última estratégia permite desligar máquinas que não são necessárias e assim poupar ainda mais energia. Este trabalho considera que as máquinas começam todas desligadas e vão sendo ligadas de acordo com a necessidade do trabalho executado.

De um modo geral, o EASAHE está dividido em duas fases. Numa primeira fase vai selecionar todas as tarefas que estão prontas a ser executas, e ordena essa lista de acordo com a comparação apresentada em 1. De seguida, percorre a lista ordenada de tarefas e escolhe uma máquina para executar a tarefa, como apresentado no algoritmo 2.

Olhando em pormenor para o algoritmo 1, o EASAHE prioriza tarefas que libertam dependências (linhas 1-5), de acordo com a pontuação de cada tarefa calculada pela equação 1, o que permite ter mais tarefas disponíveis para ocupar os recursos. Além disso, utiliza o tempo de execução previsto de cada tarefa como critério secundário para a ordenação, ou seja, vai executar primeiro tarefas mais longas (linhas 7-10) de forma a poder utilizar as tarefas mais pequenas para ocupar cores que ficam livres nas fases finais do algoritmo.

Passando agora para o algoritmo 2, onde é feita a escolha da máquina onde a tarefa vai ser executada, o EASAHE evita manter mais recursos ligados do que aqueles que são necessários para executar o trabalho, de forma a manter uma taxa de utilização dos recursos alta. A estratégia utilizada aqui consiste em verificar se o número de tarefas sem dependências que estão a espera para serem agendadas é inferior ao número de recursos que estão disponíveis (linha 5). Esta verificação pode ser melhorada com um algoritmo mais preciso, no entanto esta simples condição oferece resultados satisfatórios sem grandes impactos na complexidade do algoritmo.

Para manter as máquinas com taxas de utilização equilibradas, o EASAHE procura qual a máquina cujo tempo em que está subutilizada mais se aproxima do tempo de execução previsto da tarefa (linhas 10-11). Caso esta máquina não esteja disponível, irá verificar se a máquina que estamos a analisar consegue executar esta tarefa, de forma a que o tempo em que está subutilizada não seja maior do que o tempo de execução previsto da tarefa (linhas 13-14).

Uma vez que a escolha da melhor máquina é um cálculo computacionalmente pesado, este será apenas usado quando é determinado que a carga de trabalho disponível não é suficiente para manter as máquinas todas ocupadas. Assim, consegue-se manter os cores ocupados sem aumentar muito a complexidade do algoritmo.

É de realçar que o cálculo da melhor máquina não necessita de ser refeita a cada iteração do ciclo, uma vez que essa escolha irá manter-se até que alguma

tarefa seja agendada ou até que alguma tarefa termine de executar, o que não acontece dentro deste ciclo.

A utilização de um ciclo para percorrer todas as máquinas poderá ser útil caso seja pretendido aumentar a complexidade do algoritmo para, por exemplo, na condição onde verifica se existem tarefas suficientes para manter as máquinas ocupadas, prever se a máquina em questão irá estar ocupada no futuro com tarefas que sejam libertadas por outras tarefas que já estão a correr.

De forma a impedir que uma máquina seja ligada para ficar subutilizada, o EASAHE verifica se tem tarefas suficientes para manter a máquina ocupada (linha 28). Desta forma, se tiver apenas uma tarefa que está à espera de ser executada, não liga uma nova máquina apenas para essa tarefa, o que permite poupar o gasto energético de ter mais uma máquina ligada, além de que pode haver alguma máquina no futuro onde essa tarefa possa ser executada sem aumentar o tempo em que a máquina está subutilizada, reduzindo ainda mais o consumo energético.

Por fim, de forma a não manter máquinas ligados sem fazer nada, o EASAHE desliga as máquinas sempre que não estejam a executar nenhuma tarefa. Para isto, apenas verifica se a máquina está a executar alguma tarefa, sempre que alguma tarefa nessa máquina termina.

**Algorithm 1** Comparação entre duas tarefas para ordenar a lista de tarefas

**Input:** $t_1$ and $t_2$ tasks
**Output:** True if $t_1$ should be executed before $t_2$, and False otherwise

```
1: if getTaskScore(t1) > 0 and getTaskScore(t2) ≤ 0 then
2:     return true
3: else
4:     if getTaskScore(t2) > 0 and getTaskScore(t1) ≤ 0 then
5:         return false
6:     else
7:         if predictTime(t1) == predictTime(t2) then
8:             return t1 < t2
9:         else
10:            return predictTime(t1) > predictTime(t2)
11:        end if
12:    end if
13: end if
```

$$Score(t) = num\_childs(t) - \sum_{child \in childs(t)} num\_dependencies(t) - 1 \quad (1)$$

A equação 1 calcula a pontuação de cada tarefa tendo em conta a quantidade de dependências que são libertadas. Desta forma, o valor será positivo se

o número de tarefas libertadas após executar a tarefa atual for maior do que o número de dependências que essas tarefas libertadas ainda tem para executar, e negativo caso contrário.

**Algorithm 2** Escolha da máquina para uma dada tarefa

**Input:** $t$ a task to schedule
**Output:** The VM to execute the task $t$

```
1: available_vms ← list of available VMs
2: underutilized_vms ← list of underutilized vms
3: while underutilized_vms is not empty do
4:    vm ← next vm on the list of VMs
5:    if getNumberRemainingTasks() ≤ getNumberAvailableCores() then
6:       if task t is a priority then
7:          return vm
8:       end if
9:       best_fit ← best fit for task t on vm list available_vms
10:      if best_fit has available cores then
11:         return best_fit
12:      end if
13:      if predictTime(t) ≤ getIdleTime(vm) then
14:         return vm
15:      end if
16:   else
17:      return vm
18:   end if
19: end while
20: if underutilized_vms is empty then
21:   vm ← get VM down
22:   if no vm then
23:      if no idle hosts then
24:         return None
25:      end if
26:      while list of hosts is not empty do
27:         host ← next host on the list of hosts
28:         if host is turned off and (number of remaining tasks ≥ number of cores on
            + host.cores or no vms turned on) then
29:            return None
30:         end if
31:      end while
32:      vm ← create vm on new host
33:   end if
34:   if no vm then
35:      return None
36:   end if
37:   turn on VM vm
38:   return vm
39: end if
```

# 4 Simulador

Para desenvolver e testar o algoritmo, decidimos utilizar um simulador, uma vez que este oferece maior controlo e é mais fácil de utilizar comparativamente a um ambiente real. Assim, dentro dos diversos simuladores disponíveis, foi necessário encontrar um que permitisse uma implementação simples do algoritmo, e também que fosse capaz de simular o consumo energético de uma determinada plataforma com uma determinada carga de trabalho, ou que pelo menos permita adicionar essa análise.

Sendo assim, decidimos utilizar o simulador WRENCH [4] uma vez que possui todas as características que procuramos, além de possuir bastante suporte relativamente a trabalhos já gerados para este projeto, mas também ferramentas que permitem analisar o trabalho simulado. Uma destas ferramentas é uma *dashboard* que nos permite observar em qual máquina cada tarefa foi executada, permitindo assim verificar não só que as máquinas permaneceram ligadas ao longo da execução dos trabalhos, mas também que elas estão a ser utilizadas ao máximo.

Este simulador funciona como uma camada de abstração para a ferramenta de análise Simgrid [3]. O seu funcionamento é bastante simples, na medida em que basta fornecer um ficheiro com a descrição do trabalho a simular, outro com a descrição da plataforma que vamos utilizar, e por fim definir o algoritmo de agendamento. Este simulador permite ainda ajustar a frequência das máquinas, de acordo com a especificação da plataforma fornecida, além de permitir ligar e desligar máquinas quando necessário, o que nos permite explorar técnicas de ajuste de frequência.

A forma como o consumo energético é medido no simulador será discutido na subsecção 4.1.

## 4.1 Medição de energia

O simulador escolhido já possui suporte para o modelo energético do Simgrid, que está explicado em detalhe em [5]. No entanto, após correr alguns testes verificamos que o consumo energético por utilização de *core* eram bastante diferentes do valor real da plataforma que estávamos a simular.

Desta forma, para a calibrar a medição da energia realizámos um teste com um conjunto de trabalhos, em que cada um continha uma tarefa para o número de cores que queríamos analisar, e deixamos a correr por 1000s. De seguida, corremos um teste de stress no CPU físico para cada número de cores, com o comando *stress*, e verificamos o consumo energético com a ferramenta PowerJoular [13]. Os resultados obtidos podem ser vistos na figura 1. Ambos os testes assumem que o CPU é utilizado a 100%. Os valores utilizados para o plugin de energia do Simgrid foram: *Off* a 0 Watts, *Idle* a 0.5 Watts, *Epsilon* a 3 Watts, e *AllCores* a 70 Watts.

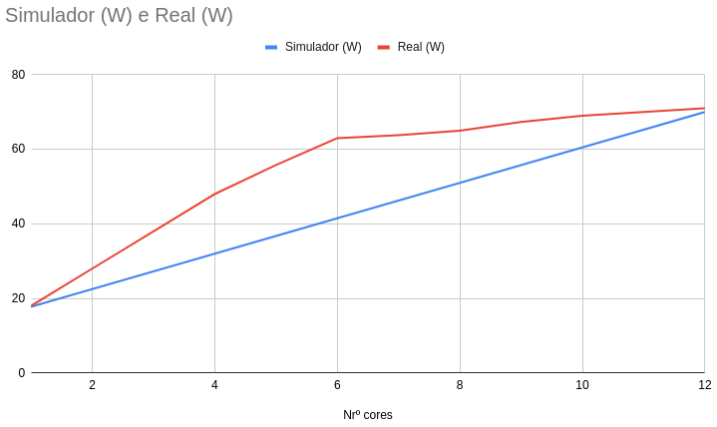


**Figura 1.** Comparação entre os resultados do simulador e os resultados reais

Como podemos observar na figura 1, o consumo energético no simulador foi linear ao número de cores utilizados, enquanto na realidade é mais ou menos linear até metade do número de cores, e após isso o consumo aumenta cada vez menos com o aumento do número de cores, o que gera uma grande diferença no resultado final. Além disso, ao utilizar o modelo do Simgrid, não iríamos verificar grandes diferenças entre utilizar metade dos cores de dois processadores ou a totalidade dos cores de um único processador.

Portando, decidimos criar no simulador um novo serviço que verifica a cada segundo quantos cores estão a ser utilizados e calculamos o consumo somando o consumo estático com o consumo dinâmico para aquele número de cores obtido no teste anterior. O cálculo pode ser observado na fórmula 2.

$$E(t) = \sum_{executor_i} Static\_Power(i,t) + Dynamic\_Power(i,t) \tag{2}$$

Os valores obtidos nos nossos testes de stress estão presentes na tabela 1

| Nrº de cores | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total (w) | 0.5 | 18 | 28 | 38 | 48 | 55.8 | 63 | 63.8 | 65 | 67.3 | 69 | 70 | 71 |

**Tabela 1.** Medições reais do consumo energético do CPU

## 5 Resultados e avaliação

Para avaliar o algoritmo proposto decidimos usar como termo de comparação os seguintes parâmetros:

– Consumo energético
– Tempo de execução
– Tempo gasto no agendamento

Quanto à carga de trabalho simulada, foram utilizados *traces* de execuções reais e também sintéticas. Para obter os *traces* reais, foi utilizada a ferramenta HiBench [6] que possui diversas cargas com múltiplas características, como analisado neste trabalho [16], e corremos os seguintes trabalhos:

– Sort com uma carga de 55 GB
– Terasort com uma carga de 500 MB
– Pagerank com uma carga de 8000000 *pages*, 3 *iterations* and 16 *block_width*

Uma vez que o simulador não possui medição de energia consumida pela utilização da rede, e a plataforma em análise é homogénea, que não poderá tirar proveito das diferentes características dos trabalhos, então estas cargas apenas vão permitir testar o comportamento do algoritmo de acordo com o número e tempo de execução das tarefas.

Estes trabalhos permitiram ainda validar as medições do tempo de execução e de consumo energético do simulador. Tal como efetuado anteriormente, o consumo energético na plataforma real foi medido utilizando a ferramenta PowerJoular.

Todos os testes de execuções reais foram efetuados num *cluster* composto por três máquinas equipadas com 16 GB de memória e um processador Intel(R) Core(TM) i5-10505 CPU @ 3.20GHz com 12 *cores*. Quanto aos testes do simulador, estes foram executados numa única máquina deste *cluster*.

| | Real (s) | Simulador (s) |
|---|---|---|
| Sort | 98.923 | 89.8234 |
| Terasort | 242.853 | 233.355 |
| PageRank | 971.254 | 975.862 |

**Tabela 2.** Comparação do tempo de execução entre o simulador e a plataforma real

| | Real (Wh) | Simulador (Wh) |
|---|---|---|
| Sort | 2.9894 | 2.35233 |
| Terasort | 6.4454 | 4.62575 |
| PageRank | 31.9905 | 28.7597 |

**Tabela 3.** Comparação do consumo energético entre o simulador e a plataforma real

Como podemos observar nestes resultados, Tabelas 2 e 3, apesar de existir uma ligeira diferença tanto no consumo energético como no tempo de execução,

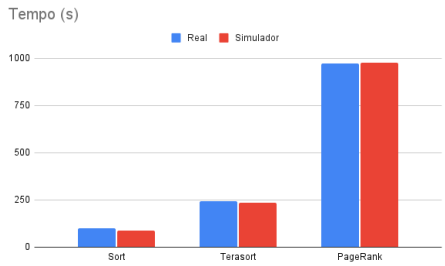


**Figura 2.** Tempo de execução

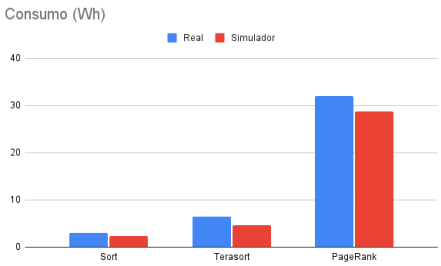


**Figura 3.** Consumo energético

estes crescem de forma bastante semelhante, comprovando a calibração do simulador e a capacidade de fazer comparações justas.

De forma a analisar como o algoritmo se comporta com um trabalho heterogéneo, ou seja, com múltiplas tarefas com diferentes características, decidimos misturar todas as tarefas das cargas anteriores e criar um novo trabalho chamado de **all-in-one**. Este trabalho revelou-se bastante interessante, uma vez que na execução com o algoritmo FIFO, ele apresentou uma utilização bastante fraca dos recursos, existindo várias situações onde múltiplos processadores apenas estavam a ser utilizados a metade da capacidade.

Além disso, utilizamos também uma carga sintética obtida em [1]. Este repositório contém um conjunto de cargas criadas para o simulador que está a ser utilizado, pelo que decidimos utilizar as seguintes cargas:

- 1000genome-chameleon-10ch-100k
- 1000genome-chameleon-20ch-250k

Como termo de comparação, decidimos utilizar o algoritmo E-FIFO, uma vez que este é uma versão do algoritmo FIFO, que está disponível nas ferramentas de processamento de dados, com atenção à eficiência energética, na medida em que desliga máquinas que não estão a ser utilizadas.

Desta forma, os resultados obtidos foram os seguintes:

| | E-FIFO (Wh) | EASAHE (Wh) | Melhoria (%) |
|---|---|---|---|
| Sort | 2.3523 | 2.33614 | 0.69 |
| Terasort | 4.6257 | 4.57278 | 1.14 |
| PageRank | 28.7598 | 28.1071 | 2.27 |
| All-in-one | 35.5408 | 31.4568 | 11.49 |
| Genome_10ch_100k | 40256.2 | 40181.4 | 0.19 |
| Genome_20ch_250k | 127818 | 127191 | 0.49 |

**Tabela 4.** Comparação consumo energético

| | E-FIFO (s) | EASAHE (s) | Melhoria (%) |
|---|---|---|---|
| Sort | 89.8234 | 87.7764 | 2.28 |
| Terasort | 233.355 | 233.658 | -0.13 |
| PageRank | 975.862 | 1207.21 | -23.71 |
| All-in-one | 1073.16 | 956.059 | 10.91 |
| Genome_10ch_100k | 713011 | 703196 | 1.38 |
| Genome_20ch_250k | 2327280 | 2444170 | -5.02 |

**Tabela 5.** Comparação tempo execução

| | E-FIFO (ms) | EASAHE (ms) | Aumento (%) |
|---|---|---|---|
| Sort | 2038.01 | 4893.1 | 140.09 |
| Terasort | 2032.61 | 4911.33 | 141.63 |
| PageRank | 17.8396 | 198.16 | 1010.79 |
| All-in-one | 11397.1 | 32479.6 | 184.98 |
| Genome_10ch_100k | 788.124 | 1127.13 | 43.01 |
| Genome_20ch_250k | 13499.4 | 20675.6 | 53.16 |

**Tabela 6.** Comparação tempo de agendamento

Ao observar os resultados da tabela 4, podemos observar que tal como previsto, o algoritmo apresenta os melhores resultados quando existe uma grande heterogeneidade nas tarefas, onde conseguiu uma melhoria de 11.49%, comprovando que consegue tirar proveito das diferenças de tempo de execução de cada uma das tarefas para utilizar os recursos ao máximo. Apesar de não apresentar melhorias tão altas nas outras cargas, o algoritmo conseguiu melhorar um pouco o custo energético. No caso do PageRank houve uma melhoria maior do que nos restantes casos porque o algoritmo determinou que das três máquinas que estavam disponíveis, apenas duas seriam necessárias para executar as tarefas, o que resulta numa poupança significativa. Quanto às restantes, principalmente o Sort e o Terasort, a melhoria foi bastante baixa porque não existem muitos recursos subutilizados, não permitindo assim ao algoritmo reorganizar as tarefas de forma mais eficiente.

Quando aos resultados presentes na tabela 5, podemos observar que apesar do consumo energético melhorar em todas as cargas, o tempo de execução varia bastante. Novamente, o all-in-one melhorou devido ao melhor aproveitamento dos recursos. Já o PageRank, apesar de melhorar um pouco no consumo energético, piora bastante no tempo de execução, devido a utilizar uma máquina a menos.

Por fim, temos o tempo gasto no agendamento das tarefas na tabela 6. De um modo geral, o EASAHE apresenta uma performance bastante boa mesmo comparado a um algoritmo tão simples como o FIFO, sendo a única exceção

foi o PageRank. Este aumento enorme pode ser explicado por o algoritmo ter feito bastantes mais iterações do que no FIFO, além de ter entrado muitas vezes nos cálculos mais pesados. De realçar que o tempo de agendamento mesmo no pior caso, é 2 ordens de grandeza inferior ao tempo de execução de qualquer das cargas, sendo por isso negligenciável.

Nas figuras 4 e 5, temos uma representação gráfica da distribuição das tarefas da carga all-in-one feitas pelos algoritmos FIFO e EASAHE respetivamente. Ao analisar estes figuras, é possível concluir que o EASAHE consegue alterar a ordem das tarefas de forma a equilibrar as tarefas de forma muito mais eficaz do que o FIFO, conseguindo assim aproveitar melhor os recursos disponíveis.

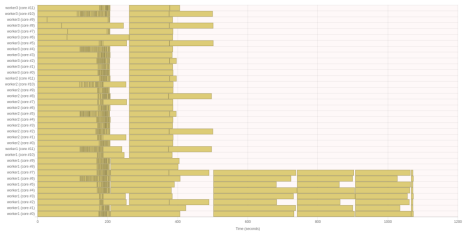

**Figura 4.** all-in-one com o FIFO

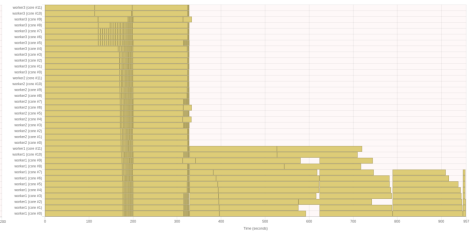

**Figura 5.** all-in-one com o EASAHE

## 6 Conclusão e trabalho futuro

Apesar do recente desenvolvimento de algoritmos para o agendamento de trabalhos em ferramentas de processamento de dados com preocupações de eficiência energética, estes foram desenvolvidos para funcionar em ambientes heterogéneos, de forma a tirar o máximo partido das diferenças entre as máquinas. No entanto, tipicamente os *clusters* de processamento de dados possuem máquinas homogéneas.

Neste artigo propomos o EASAHE, um algoritmo de agendamento que consegue melhorar o consumo energético em ambientes homogéneos. A avaliação usando *traces* de execuções reais e sintéticas em Spark, revela que melhora a eficiência energética de forma consistente e melhora o tempo de execução quando a carga possui uma grande variedade de tarefas.

Como trabalho futuro, uma vez que o algoritmo consegue manter os recursos ocupados de forma eficaz e até reduzir o tempo de execução, torna-se um bom candidato para aplicar técnicas de ajuste de frequência, de forma a atrasar algumas tarefas para diminuir mais o consumo energético. Além disso, será interessante analisar o impacto do algoritmo no consumo energético da rede. Por fim, também será possível analisar outras técnicas mais complexas dentro do algoritmo de forma a criar um compromisso entre consumo energético e complexidade.

## Referências


1. Pegasus Workflow Execution Instances (May 2023), https://github.com/wfcommons/pegasus-instances, original-date: 2016-11-27T05:14:05Z
2. Berral, J.L., Goiri, , Nou, R., Julià, F., Guitart, J., Gavaldà, R., Torres, J.: Towards energy-aware scheduling in data centers using machine learning. In: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. pp. 215–224. e-Energy '10, Association for Computing Machinery, New York, NY, USA (Apr 2010). https://doi.org/10.1145/1791314.1791349, https://doi.org/10.1145/1791314.1791349
3. Casanova, H.: Simgrid: a toolkit for the simulation of application scheduling. In: Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 430–437 (2001). https://doi.org/10.1109/CCGRID.2001.923223
4. Casanova, H., Ferreira da Silva, R., Tanaka, R., Pandey, S., Jethwani, G., Koch, W., Albrecht, S., Oeth, J., Suter, F.: Developing Accurate and Scalable Simulators of Production Workflow Management Systems with WRENCH (Nov 2020). https://doi.org/10.1016/j.future.2020.05.030, https://github.com/wrench-project/wrench
5. Heinrich, F.C., Cornebize, T., Degomme, A., Legrand, A., Carpen-Amarie, A., Hunold, S., Orgerie, A.C., Quinson, M.: Predicting the energy-consumption of mpi applications at scale using only a single node. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 92–102 (2017). https://doi.org/10.1109/CLUSTER.2017.66
6. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010). pp. 41–51 (Mar 2010). https://doi.org/10.1109/ICDEW.2010.5452747
7. Jianchao, T., Shuqiang, Y., Chaoqiang, H., Zhou, Y.: Design and Implementation of Scheduling Pool Scheduling Algorithm Based on Reuse of Jobs in Spark. In: 2016 IEEE First International Conference on Data Science in Cyberspace (DSC). pp. 290–295 (Jun 2016). https://doi.org/10.1109/DSC.2016.81
8. Kocot, B., Czarnul, P., Proficz, J.: Energy-aware scheduling for high-performance computing systems: A survey. Energies **16**(2) (2023). https://doi.org/10.3390/en16020890, https://www.mdpi.com/1996-1073/16/2/890
9. Li, H., Wang, H., Fang, S., Zou, Y., Tian, W.: An energy-aware scheduling algorithm for big data applications in Spark. Cluster Computing **23**(2), 593–609 (Jun 2020). https://doi.org/10.1007/s10586-019-02947-9, https://doi.org/10.1007/s10586-019-02947-9
10. Li, H., Wei, Y., Xiong, Y., Ma, E., Tian, W.: A frequency-aware and energy-saving strategy based on dvfs for spark. The Journal of Supercomputing **77** (10 2021). https://doi.org/10.1007/s11227-021-03740-5
11. Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., Meng, Z., Alizadeh, M.: Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM Special Interest Group on Data Communication. pp. 270–288. SIGCOMM '19, Association for Computing Machinery, New York, NY, USA (Aug 2019). https://doi.org/10.1145/3341302.3342080, https://doi.org/10.1145/3341302.3342080
12. Mämmelä, O., Majanen, M., Basmadjian, R., Meer, H., Giesler, A., Homberg, W.: Energy-aware job scheduler for high-performance computing. Computer Science

- Research and Development **27** (11 2012). https://doi.org/10.1007/s00450-011-0189-6
13. Noureddine, A.: PowerJoular and JoularJX: Multi-Platform Software Power Monitoring Tools. In: 2022 18th International Conference on Intelligent Environments (IE). pp. 1–4. IEEE, Biarritz, France (Jun 2022). https://doi.org/10.1109/IE54923.2022.9826760, https://ieeexplore.ieee.org/document/9826760/
14. Pinto, G., Castor, F.: Energy efficiency: A new concern for application software developers. Commun. ACM **60**(12), 68–75 (nov 2017). https://doi.org/10.1145/3154384, https://doi.org/10.1145/3154384
15. Salinas-Hilburg, J., Zapater, M., Moya, J., Ayala, J.: Energy-aware task scheduling in data centers using an application signature. Computers Electrical Engineering **97** (12 2021). https://doi.org/10.1016/j.compeleceng.2021.107630
16. Shi, W., Li, H., Guan, J., Zeng, H., jahan, R.M.: Energy-efficient scheduling algorithms based on task clustering in heterogeneous spark clusters. Parallel Computing **112**, 102947 (Sep 2022). https://doi.org/10.1016/j.parco.2022.102947, https://www.sciencedirect.com/science/article/pii/S0167819122000436
17. Shi, W., Li, H., Zeng, H.: DRL-based and Bsld-Aware Job Scheduling for Apache Spark Cluster in Hybrid Cloud Computing Environments. Journal of Grid Computing **20**(4), 44 (Dec 2022). https://doi.org/10.1007/s10723-022-09630-1, https://link.springer.com/10.1007/s10723-022-09630-1
18. Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S., Li, K.: An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. Journal of Grid Computing **14** (Apr 2015). https://doi.org/10.1007/s10723-015-9334-y
19. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems **13**(3), 260–274 (2002). https://doi.org/10.1109/71.993206
20. Ullman, J.D.: NP-complete scheduling problems. Journal of Computer and System Sciences **10**(3), 384–393 (Jun 1975). https://doi.org/10.1016/S0022-0000(75)80008-0, https://www.sciencedirect.com/science/article/pii/S0022000075800080
21. Wang, L., Tao, J., von Laszewski, G., Chen, D.: Power Aware Scheduling for Parallel Tasks via Task Clustering. In: 2010 IEEE 16th International Conference on Parallel and Distributed Systems. pp. 629–634 (Dec 2010). https://doi.org/10.1109/ICPADS.2010.128, iSSN: 1521-9097
22. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache Spark: a unified engine for big data processing. Communications of the ACM **59**(11), 56–65 (Oct 2016). https://doi.org/10.1145/2934664, https://doi.org/10.1145/2934664
23. Zhao, S.: A FIFO Spin-based Resource Control Framework for Symmetric Multiprocessing https://core.ac.uk/reader/160470220