1E1-R-1-1

クラウド環境で実行する探索アルゴリズムにおける反復資源要求法

Iterative Resource Allocation for Cloud-Based Search Algorithms

福永アレックス[*1] Alex Fukunaga　岸本 章宏[*2] Akihiro Kishimoto　Adi Botea[*3]

[*1] 東京大学 The University of Tokyo　[*2] 東京工業大学 Tokyo Institute of Technology　[*3] IBM Research, Dublin

Ａ＊系の探索アルゴリズムは必要なメモリ領域が深刻な問題であると従来から考えられているが、クラウドの普及により、膨大なメモリ資源が利用可能になった。一方、計算機資源を使用すれば課金されるので、クラウド上の探索アルゴリズムは金銭コストを抑えつつ、高い解答能力を保つ必要がある。本研究では反復的に計算機資源を要求する手法（ＩＡ）を提案し、ＩＡの有効性を理論的及び実験的に実証する。

1. Introduction

[Note: This paper is a summary of [Fukunaga 12].]

Cloud computing resources such as Amazon EC2, which offer computational resources on demand, have become widely available in recent years. In addition to cloud computing platforms, there is an increasing availability of massively parallel, high-performance computing (HPC) clusters. These large-scale *utility computing resources* share two characteristics that have significant implications for parallel search algorithms. First, vast (practically unlimited) aggregate, memory and CPU resources are available on demand. Secondly, resource usage incurs a direct monetary cost.

Previous work on parallel search algorithms has focused on *makespan*: minimizing the runtime (wall-clock time) to find a solution, given fixed hardware resources; and *scalability*: as resource usage is increased, how are makespan and related metrics affected? However, the availability of virtually unlimited resources at some cost introduces a new context for parallel search algorithm research where an explicit consideration of cost-performance tradeoffs is necessary. For scalable algorithms, it is possible to reduce the makespan by allocating more resources (up to some point). In practice, this incurs a high cost with diminishing marginal returns. For parallel A* variants, under-allocating resources results in memory exhaustion. On the other hand, over-allocation is costly and undesirable.

We consider cost-efficient strategies for dynamically allocating utility computing resources. We propose and analyze iterative allocation, a simple strategy that repeatedly runs a search algorithm with increasing resources until the problem is solved (Section 3.). Bounds on the cost incurred by iterative allocation, compared to the optimal cost, are derived (Section 4.). For a realistic class of utility computing environments and search problems, the cost suboptimality of our policy is bounded by a constant factor as small as 4. That is, we will never pay more than 4 times the a priori unknown optimal price. We validate our analysis experimentally by applying iterative allocation to the HDA* [Kishimoto 09] algorithm (Section 5.). Results on classical planning and multiple sequence alignment problems, run on 3 distinct, massively parallel computing environments, indicate that the costs incurred by iterative allocation are reasonably close to optimal, and significantly better than the worst-case upper bounds.

連絡先: Alex Fukunaga, fukunaga@idea.c.u-tokyo.ac.jp

2. Utility Computing Services

There are several types of utility computing services, including clouds, grids, and shared, massively parallel clusters. In all of these utility computing services, there is some notion of an atomic (discrete) unit of resource usage.

Definition 1 (Hardware Allocation Unit). *A hardware allocation unit (HAU), is the minimal, discrete resource unit that can be requested from a utility computing service.It is characterized by a specific number of CPU cores and a given amount of RAM, e.g., 4 cores and 8GB.*

Various HAU types can be available, each with different performance characteristics and cost.

Typical cloud services such as EC2 provision HAUs to the user immediately upon request – there is a delay, usually within 2 minutes, while the user's VM image is allocated, loaded and booted [Iosup 11]. Usage charges apply from the time that the allocated VM enters a "running" state to when it is stopped/terminated, regardless of the portion of time spent on actual computations. HAUs can be dynamically added to/removed from a running cluster.

In contrast, in HPC clusters, users typically submit *jobs* to a centralized scheduler (resource manager), where a job is a request (script) specifying an executable program and the amount of resources to use. The scheduler decides when the submitted job actually executes [Feitelson 97]. Usage charges apply for the time consumed by the user's job.

In a *continuous cost model*, the cost of resource usage is a linear function of the amount of resources used. Batch job based systems such as typical grid and shared cluster environments usually adopt a continuous cost model. On the other hand, currently, 3 of the largest commercial cloud service providers (Amazon EC2, Windows Azure, Google App Engine) all apply a *discrete cost model* in which all charges are per "HAU hour". Usage of a HAU for any fraction of an hour is rounded up, e.g., a 1 hour, 1 second allocation of a HAU which costs \$0.68/hr will cost \$1.36.

3. Iterative Allocation

A scalable, *ravenous algorithm* is an algorithm (1) which can be executed on an arbitrary number of processors, and (2) whose memory consumption continually increases until either a solution is found, or the algorithm terminates (fails) due to memory exhaustion. For example, HDA* [Kishimoto 09], a recent parallel variant of A*, is a scalable, ravenous algorithm.

Algorithm 1 Generic, Iterative Allocation (IA)

```
numHAUs ← 1
while true do
  result ← Run algorithm $a$ on problem $p$
  if result = solved then
    return success
  else
    numHAUs ← Increase(numHAUs)
```

The *iterative allocation* (IA) strategy outlined in Algorithm 1 repeatedly runs a ravenous algorithm a until the given problem is solved. This is very simple, but the key detail is the `Increase()` function, which decides the number of HAUs to allocate in the next iteration. We seek a policy for choosing the number of HAUs allocated on each iteration of IA which tries to minimize the total cost.

3.1 Analysis of IA: Preliminaries

We present formal properties of the generic allocation policy. For formal analysis, we make two assumptions.

Assumption 1 (Homogeneous hardware allocation units)**.** *All HAUs used by IA are identical hardware configurations.*

Assumption 2 (Monotonicity)**.** *If a problem is solved on i HAUs, then it will be solved on $j > i$ HAUs.*

The cost to solve a problem is defined in terms of HAU-hours. When the problem at hand can be solved on v HAUs, let T_v be the makespan time needed to solve the problem. In a *continuous cost model*, common in shared HPC clusters,the cost of solving the problem on v HAUs is defined as $T_v \times v$.[*1] In a *discrete cost model*, common among commercial cloud services, the cost is $\lceil T_v \rceil \times v$. In the rest of the paper, unless the cost model (continuous vs discrete) is explicitly stated or clear from the context, our statements apply to both models.

Definition 2. *The* minimal width *W^+ is the minimum number of HAUs that can solve a problem with a given ravenous algorithm. The cost incurred by using W^+ HAUs is denoted C^+.*

Definition 3. *Given a cost model, the* minimal cost width *W^* is the number of HAUs that results in a minimal cost. When several width values achieve the minimal cost, W^* refers to the smallest witdh with such a property. The minimal cost (i.e., the cost to solve the problem with W^* HAUs in use) is written as C^*.*

*1 Without loss of generality, we assume the cost per HAU-hour to be 1, slightly simplifying the equations.

If W^* is known a priori, a cost-optimal strategy for solving the problem at hand is trivial: rent W^* HAUs in parallel until the problem is solved. Most of time, however, W^+ or W^* are not known a priori. The best we can hope for is to develop strategies that approximate the optimal values.

Definition 4 (Nonincreasing search efficiency)**.** *Given a ravenous algorithm, we say that the search efficiency is nonincreasing if the following two conditions hold: (1) If n_v is the number states generated using v HAUs, then $n_v \leq n_{v+1}, \forall v \geq W^+$, and (2) $T_v v \leq T_{v+1}(v+1), \forall v \geq W^+$.*

As the number of HAUs increases, search efficiency typically decreases because of factors such as an increased search overhead and communication overhead.

Proposition 1. *In a continuous cost model, if the search efficiency is nonincreasing, $C^* = C^+$ and $W^* = W^+$.*

In a discrete-cost model, the min-width cost C^+ is not necessarily the same as the minimal cost C^*. For example, suppose that running a ravenous algorithm using 1 HAU will exhaust memory, 2 HAUs can solve the problem in 1.3 hours (which is rounded up to 2 hours), and 3 HAUs can solve the problem in 1 hour. In this case, the min-width is $W^+ = 2$, $C^+ = 4$, but the min-cost $C^* = 3$. However, the gap between C^* and C^+ is relatively small:

Proposition 2. *In a discrete cost model, if the search efficiency is nonincreasing, $C^+ < C^* + W^+$.*

Later, we shall see that under realistic conditions, $W^* = W^+$ and $C^* = C^+$ in a discrete cost model, regardless of whether search efficiency is nonincreasing (Sec. 4.1).

Definition 5. *The* max iteration time *E is the maximum actual (not rounded up) time that an iteration can execute before at least 1 HAU exhausts memory and fails.*

The previous cost definitions are based on a fixed number of HAUs. On the other hand, IA varies the number of HAUs dynamically. In a continuous cost model, assuming that IA allocates v_i HAUs at iteration i, the cost incurred by IA is $I = \sum_{i=0}^{j} D_{v_i} v_i$. D_{v_i}, with $i < j$, is the time taken to fail for all but the last iteration. $D_{v_j} = s_{v_j}$ is the time to successfully complete the last iteration. In all cases, $D_{v_i} \leq E$. In a discrete cost model, times spent by individual HAUs are rounded up. HAUs will use any spare time left at the end of one iteration to start the next iteration. Thus, for each HAU used in solving a problem, the total time spent by that HAU across all iterations where it participated is rounded up. The next example clarifies this further.

Example 1 (Cost computation in discrete cost model)**.** *A doubling IA is executed: iteration 0 uses 1 HAU, iteration 1 uses 2 HAUs, iteration 2 uses 4 HAUs. In other words,* *one HAU is continuously used in all 3 iterations. One HAU is used in the last two iterations, and two HAUs participate only to the last iteration. The cost is \$1 per HAU-hour. Each failed iteration requires .33 hours. The last, successful iteration requires .25 hours. The total cost is:* $1\ HAU \times \lceil .33 + .33 + .25 \rceil\ hours \times \$1 + 1\ HAU \times \lceil .33 + .25 \rceil\ hrs \times \$1 + 2\ HAU \times \lceil .25 \rceil\ hrs \times \$1 = 1 + 1 + 2 = \$4$.

Definition 6 (Min-width cost ratio of a strategy). *The min-width cost ratio R^+ is defined as $I(S)/C^+$, where $I(S)$ is the cost of IA (using a particular allocation strategy S) until the problem is solved, and C^+ is the min-width cost.*

Definition 7 (Min-cost ratio of a strategy). *The min-cost ratio R^* is defined as $I(S)/C^*$, where C^* is the min. cost.*

4. The Geometric (b^i) Strategy

Consider a simple strategy where the number of HAUs allocated on the i-th iteration is $\lceil b^i \rceil$, for some $b > 1$. For example, the 2^i (doubling) strategy doubles the number of HAUs allocated on each iteration: First, try 1 HAU; if it fails, try 2 HAUs, then 4 HAUs, and so on.

Suppose the b^i strategy solves a problem on the j-th iteration, i.e., $j = \lceil log_b W^+ \rceil$, where W^+ is the min width.

The cost of a geometric allocation strategy with a continuous cost model is $I = \sum_{i=0}^{j} D_{b^i} b^i$. Recall that $D_{b^j} = s_{b^j} \leq E$ on the successful iteration, and $D_{b^i} \leq E$ on the failed iterations. By standard manipulations, $I \leq E\frac{b^j-1}{b-1} + s_{b^j} b^j$. In the discrete case, $I \leq \lceil E \rceil \frac{b^j-1}{b-1} + \lceil s_{b^j} \rceil b^j$. This latter upper bound is obtained by making the pessimistic relaxation that no spare time is used by a HAU from one iteration to the next. This explains in part why our experimentally measured cost ratios are better than the theoretical upper bounds introduced later in this section.

We specialize the cost ratio analysis to a class of realistic cloud environments and ravenous search algorithms.

4.1 Discrete Cost Model with Fast Memory Exhaustion

Cloud platforms such as Amazon EC2 and Windows Azure typically have discrete cost models, where the discrete billing unit is 1 hour, and fractional hours are rounded up. This relatively long unit of time, combined with the fast rate at which search algorithms consume RAM, has significant practical implications:

Observation 1. *Both the success time and the failure time are at most 1 billing time unit (1 hour), $D_k \leq E \leq 1, \forall k \geq 1$. A direct consequence is that $\lceil D_k \rceil = \lceil E \rceil = 1, \forall k \geq 1$.*

Currently, the amount of RAM per core on EC2 and Windows Azure ranges from 2 to 8.5 GB per core. Some of the RAM in each core must be used for purposes other than search state storage, e.g., the OS, MPI communication queue, and heuristic tables such as pattern databases. Assume (optimistically) that we have 8 GB RAM remaining after this "overhead" is accounted for. Suppose that a state requires 50 bytes of storage overall in the system. Generating at least 46,000 new states per second, which is a relatively slow state generation rate, will exhaust 8 GB RAM within an hour. Many search applications generate states much faster than this.

Thus, many (but not all) search applications will exhaust the RAM/core in a HAU within a single billing time unit in modern cloud environments. A single iteration of IA will either solve a given problem within 1 billing time unit, or fail (due to RAM exhaustion) within 1 billing time unit.

Our experiments (Sec. 5.) validate Observation 1 for all of our planning and sequence alignment benchmarks.

Observation 2. *In a discrete cost model with $E \leq 1$, the cost to solve a problem on v HAUs is proportional to v. As a direct consequence, $W^+ = W^*$ and thus $R^+ = R^*$.*

Remarkably, Observation 2 holds independently of whether the search efficiency is nonincreasing (Definition 4). Although we used Def. 4 to obtain general case results (Propositions 1 and 2), under the stronger condition that $E \leq 1$, all of the results below hold regardless of whether the search efficiency is nonincreasing.

The cost overhead of IA consists of two components: (1) unnecessary HAUs allocated on the final, successful iteration, and (2) repeated allocation of HAUs due to failed iterations. When the min-width happens to be a power of b, then the former overhead is 0. In a discrete pricing model, the latter overhead can be reduced significantly when iterations terminate faster than a single billing unit, and thus u iterations fit in $v < u$ billing units. Furthermore, with a sufficiently small E, *all iterations can be executed within a single billing time unit*, entirely eliminating the repeated allocation cost overhead. Indeed, in our experiments below, for all our planning benchmark problems, all iterations fit in a single billing time unit.

The worst case for the min-cost ratio $R^+ = R^*$, occurs when when the $(j-1)$-th iteration is barely insufficient, and on the final j-th iteration, only $W^+ = b^{j-1} + 1$ HAUs (out of the b^j allocated) are necessary:

$R^*_{wo} = R^+_{wo} = \frac{I}{\lceil s_{W^+} \rceil (b^{j-1}+1)} \leq \frac{b^2}{b-1}$

(See [Fukunaga 12] for derivation, as well as average case bound).

Observation 3. *When $E \leq 1$, the worst case bound $b^2/(b-1)$ is minimized by the doubling strategy ($b = 2$).*

As b increases above 2, making the iterative allocation more aggressive, the upper bound for R^*_{avg} improves, but the worst case gets worse. Therefore: *the simple, doubling strategy is the natural allocation policy to use in practice.*

Note that for the doubling strategy (2^i), the worst case cost ratio does not exceed 4. In other words, with the simple doubling strategy in a discrete cost model when $E \leq 1$, we will *never pay more than 4 times the optimal cost* that we would have paid if we knew the optimal width in advance.

5. Experimental Results

We experimentally evaluate iterative allocation applied to HDA* (**IAHDA***). We focus on the doubling strategy,

	# Cores & (RAM) per HAU	Max # HAUs (cores)	number of problems solved on iteration							Continuous Model Min-Width Cost Ratio (R^+)			Discrete Model Min-Cost Ratio (R^*)		
			2	3	4	5	6	7	Total	Mean	SD	Max	Mean	SD	Max
Planning: `HPC`	12(54GB)	64(768)	2	5	11	3	3	1	25	**2.18**	0.45	3.34	**1.29**	0.27	1.88
Planning: `Commodity`	8(16GB)	8(64)	5	1	2	-	-	-	8	**1.62**	0.29	2.29	**1.04**	0.12	1.33
Planning: `EC2`	4(15GB)	16(64)	6	1	1	5	-	-	13	**1.63**	0.25	2.27	**1.26**	0.25	1.64
Mult. Seq. Align: `HPC`	12(54GB)	64(768)	4	1	-	1	-	2	8	**2.02**	0.46	2.76	**1.54**	0.75	3.28

表 1: Summary of IAHDA* on planning and multiple sequence alignment on `HPC`, `Commodity`, and `EC2` clusters.

for the reasons outlined in the previous section. Domain-independent planning and multiple sequence alignment problems are solved on 3 parallel clusters: (1) `HPC` - a large-scale, high-performance computing cluster, where each HAU has 12 cores (Intel Xeon 2.93GHz), 4.5GB RAM/core, and a 40GB Infiniband network. (2) `Commodity` - a cluster of commodity machines, where each HAU has 8 cores (Xeon 2.33GHz) and 2GB RAM/core, and a 1Gbps (x3, bonded) Ethernet network. (3) `EC2` - Amazon EC2 cloud cluster using the `m1.xlarge` ("Extra Large" instance) HAU, which have 4 virtual cores, (3.75GB RAM per core, and an unspecified network interconnect).

We evaluated IAHDA* for domain-independent planning on a Fast-Downward based planner using the merge-and-shrink (M&S) heuristic [Helmert 07]. We use 7 standard benchmark domains: Depot, Driverlog, Freecell, Logistics, Mprime, Pipesworld-Notankage, Pipesworld-Tankage (142 problems total). We also evaluated IAHDA* on multiple sequence alignment (MSA) using the variant of HDA* in [Kobayashi 11], without the weighted-A* preprocessing/upper-bounding. The test set consisted of 28 standard alignment problems for 5-9 sequences (`HPC` only).

For each problem, on each cluster, the min-width W^+ was found by incrementing the number of HAUs until the problem was solved. We evaluate the data under both the continuous and discrete cost models: For both the continuous and discrete models, we computed the min-width cost ratio R^+. In the discrete model, we assume the industry standard 1 hour granularity. In all our test problems, max iteration time E (Def 5) turns out to be less than 1 hour. Thus, by Observation 2, discrete $R^* = R^+$ (the number of HAUs which minimizes cost is equal to min-width).

Table 1 summarizes the results. For all 3 clusters, we only consider problems which required ≥ 2 iterations on that cluster. For each system, we show how many problems were solved using exactly i iterations for each $2 \leq i \leq \log_2(\text{Max\#HAU's})$, as well as the total number of problems solved using ≥ 2 iterations. For both the continuous and discrete cost models, we show the mean, standard deviation, min, and max values of the min-width cost ratio R^+ or min-cost ratio R^* (see above). See [Fukunaga 12] for detailed results.

From the experimental results (Tables 1), we observe:
(a) The mean discrete min-cost ratios R^* for all problems, on all 3 clusters (Table 1) is significantly less than the theoretical worst case bound (4.0) for the doubling strategy (Sec. 4.1); The continuous min-width cost ratio R^+ was never higher than 3.34.
(b) For all our benchmarks, $E < 1$, satisfying the conditions of Sec. 4.1. On all planning problems, all iterations were performed within a single billing time unit (hour). Furthermore, on some problems, W^* is a power of 2, and the discrete $R^* = 1.0$, i.e., no additional cost was incurred by IA, compared to the optimal (minimal) cost.

6. Discussion and Conclusions

This paper explored some implications of having access to vast (but costly) resources for parallel search algorithms. We analyzed a general, iterative resource allocation strategy for scalable, memory-intensive search algorithms, including parallel A* variants such as HDA*. We presented bounds on the relative cost of a simple, geometric allocation policy, compared to an (a priori unknown) optimal allocation. Under realistic assumptions and a discrete pricing model used in commercial clouds such as Amazon EC2 and Windows Azure, we showed that the worst case cost ratios for a doubling strategy was at most 4 (the average case bound is 2.67 – see [Fukunaga 12]). Experiments with planning and sequence alignment validated the theoretical predictions, and showed that the cost ratios can be quite low, showing that IA with a doubling policy is a reasonable strategy in practice. While our experiments applied IA to HDA*, our theoretical analysis is quite general, and applies to any scalable, ravenous algorithm that satisfies the assumptions, including, for example, scalable work-stealing A*.

参考文献

[Feitelson 97] Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P.: Theory and Practice in Parallel Job Scheduling, in Feitelson, D. G. and Rudolph, L. eds., *JSSPP*, Vol. 1291 of *Lecture Notes in Computer Science*, pp. 1–34, Springer (1997)

[Fukunaga 12] Fukunaga, A., Kishimoto, A., and Botea, A.: Iterative Resource Allocation for Memory Intensive Parallel Search Algorithms on Clouds, Grids and Shared Clusters, in *Proceedings of AAAI*, p. to appear (2012)

[Helmert 07] Helmert, M., Haslum, P., and Hoffmann, J.: Flexible Abstraction Heuristics for Optimal Sequential Planning, in *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling ICAPS-07*, pp. 176–183 (2007)

[Iosup 11] Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. H. J.: Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 22, No. 6, pp. 931–945 (2011)

[Kishimoto 09] Kishimoto, A., Fukunaga, A., and Botea, A.: Scalable, Parallel Best-First Search for Optimal Sequential Planning, in *Proc. ICAPS*, pp. 201–208 (2009)

[Kobayashi 11] Kobayashi, Y., Kishimoto, A., and Watanabe, O.: Evaluations of Hash Distributed A* in Optimal Sequence Alignment, in *Proc. IJCAI*, pp. 584–590 (2011)