

Iterative Resource Allocation for Memory Intensive Parallel Search Algorithms (Extended Abstract)*

Alex Fukunaga[†]
The University of Tokyo

Akihiro Kishimoto[‡]
Tokyo Institute of Technology

Adi Botea[§]
IBM Research, Dublin, Ireland

1 Introduction

Cloud computing resources such as Amazon EC2 have become widely available in recent years. In addition, there is an increasing availability of massive-scale, distributed grid computing resources such as TeraGrid/XSEDE, and 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 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 (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? With the vast amounts of aggregate memory available in utility computing, the *monetary cost* can be the new limiting factor, since one can exhaust funds long before allocating all of the resources available to rent.

We consider cost-efficient strategies for dynamically allocating utility computing resources. We introduce an iterative allocation (IA) strategy and derive bounds on the costs incurred by IA, compared to optimal costs. For a realistic class of computing environments and search problems, the cost suboptimality is bounded by a constant multiplicative factor as small as 4. We apply IA to HDA* (Kishimoto et al. 2009), a parallel version of A* with hash-based work distribution. Results on AI planning and multiple sequence alignment, on 3 distinct, massively parallel environments, show that the IA costs are reasonably close to optimal, and significantly better than the worst-case upper bounds.

2 Utility Computing Services

In utility computing services, such as clouds, grids, and massively parallel clusters, there is some notion of an atomic

*A longer version of this paper appears in the Proceedings of AAAI 2012.

[†]This research is supported by a grant from JSPS.

[‡]This research is supported by the JST PRESTO program and KAKENHI.

[§]Part of this work was performed when this author was affiliated with NICTA and The Australian National University. Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

unit of resource usage. A *hardware allocation unit* (HAU), is the minimal, discrete resource unit that can be requested from a utility computing service. Various HAU types can be available, each with different performance characteristics and cost. Commercial clouds such as EC2 tend to have an immediate HAU allocation model with *discrete* charges. Usage of a HAU for any fraction of an hour is rounded up. Grids and shared clusters tend to be batch-job based with a *continuous* cost model. Jobs are submitted to a centralized scheduler, with no guarantees about when a job will be run. The cost is a linear function of the amount of resources used.

3 Iterative Allocation Strategy

A scalable, *ravenous algorithm* is an algorithm that can run on an arbitrary number of processors, and whose memory consumption increases as it keeps running. HDA* is an example of a scalable, ravenous algorithm.

Our *iterative allocation* (IA) strategy repeatedly runs a ravenous algorithm a until the problem is solved. The key detail is deciding the number of HAUs to allocate in the next iteration, if the previous iteration failed. We seek a policy that tries to minimize the total cost.

For formal analysis, we make two assumptions. Firstly, all HAUs used by IA are identical hardware configurations. Secondly, if a problem is solved on i HAUs, then it will be solved on $j > i$ HAUs (monotonicity). Monotonicity is usually (implicitly) assumed in the previous work on parallel search.

Let T_v be the makespan (wall-clock) time needed to solve a problem on v HAUs. In a continuous cost model, the cost on v HAUs is $T_v \times v$. In a discrete cost model, the cost is $\lceil T_v \rceil \times v$. The *minimal width* W^+ is the minimum number of HAUs that can solve a problem with a given ravenous algorithm. Given a cost model (i.e., continuous or discrete), C^+ is the associated *min width cost*. C^* is the optimal cost to solve the problem, and the *minimal cost width* W^* is the number of HAUs that results in a minimal cost. Since W^* is usually not known a priori, the best we can hope for is to develop strategies that approximate the optimal values.

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

The *min-width cost ratio* R^+ is defined as $I(S)/C^+$, where $I(S)$ is the total cost of IA (using a particular al-

	# Cores & (RAM) per HAU	Max # HAU's (cores)	number of problems solved on iteration							Continuous Model			Discrete Model			
			2	3	4	5	6	7	Total	Min-Width Cost Ratio (R^+)			Min-Cost Ratio (R^*)			# Solved w. Zero Cost Overhead
			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	8
Planning: Commodity	8(16GB)	8(64)	5	1	2	-	-	-	8	1.62	0.29	2.29	1.04	0.12	1.33	7
Planning: EC2	4(15GB)	16(64)	6	1	1	5	-	-	13	1.63	0.25	2.27	1.26	0.25	1.64	6
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	3

Table 1: Summary of IAHD A^* on planning and multiple sequence alignment on HPC, Commodity, and EC2 clusters.

location strategy S). The *min-cost ratio* R^* is defined as $I(S)/C^*$. The total cost $I(S)$ of IA is accumulated over all iterations. In a discrete cost model, times spent by individual HAUs are rounded up. The effect of the rounding up is alleviated by the fact that HAUs will use any spare time left at the end of one iteration to start the next iteration.

4 The Geometric (b^i) Strategy

The geometric strategy allocates $\lceil b^i \rceil$ HAUs at iteration i , for some $b > 1$. For example, the 2^i (doubling) strategy doubles the number of HAUs allocated on each iteration.

We focus our cost ratio analysis on a class of realistic cloud environments and ravenous search algorithms. Cloud platforms such as Amazon EC2 and Windows Azure typically have discrete cost models, where the discrete billing unit is 1 hour. This relatively long unit of time, combined with the fast rate at which search algorithms consume RAM, leads to the observation that many (but not all) search applications will exhaust the RAM/core in a HAU within a single billing time unit in modern cloud environments. In other words, a single iteration of IA will complete (by either solving the problem or running out of memory) within 1 billing time unit (i.e., $E \leq 1$). Our experiments validate this observation for all of our planning and sequence alignment benchmarks. In addition, HDA * has been observed to exhaust memory within 20 minutes on every planning and 24-puzzle problem studied in (Kishimoto et al. 2012). With a sufficiently small E , all iterations could be executed within a single billing time unit, entirely eliminating the repeated allocation cost overhead. Indeed, for all our planning benchmark problems, all iterations fit in a single billing time unit.

It is easy to see that, 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 $R^+ = R^*$. It can be shown that in the *best case*, $R^* = R^+ = 1$, in the *worst case*, $R_{wo}^* = R_{wo}^+ \leq \frac{b^2}{b-1}$, and in the average case, $R_{avg}^* = R_{avg}^+ \leq \frac{2b^2}{b^2-1}$. The worst case bound $b^2/(b-1)$ is minimized by the doubling strategy ($b = 2$). As b increases above 2, the upper bound for R_{avg}^* improves, but the worst case gets worse. Therefore: *the doubling strategy is the natural allocation policy to use in practice*. For the 2^i strategy, the average case ratio is bounded by $8/3 \approx 2.67$, and the worst case cost ratio does not exceed 4. With the 2^i strategy in a discrete cost model when $E \leq 1$, we *never pay more than 4 times the optimal, but a priori unknown cost*.

5 Experimental Results

We evaluate IA applied to HDA * (IAHD A^*), with the doubling strategy, on 3 parallel clusters: HPC - a large-scale, high-performance cluster, where each HAU has 12 cores (Intel Xeon 2.93GHz), 4.5GB RAM/core, and a 40GB Infini-band network; Commodity - a cluster of commodity machines, where each HAU has 8 cores (Xeon 2.33GHz) and 2GB RAM/core, and a 1Gbp (x3, bonded) Ethernet network; EC2 - Amazon EC2 cloud cluster using the `m1.xlarge` (“Extra Large”) HAU, which has 4 virtual cores, 3.75GB RAM per core, and an unspecified network interconnect.

We evaluated IAHD A^* for planning on a Fast-Downward based planner using the merge-and-shrink (M&S) heuristic (Helmert et al. 2007). We use 7 benchmark domains (142 problems) where the M&S heuristic is competitive with the state of the art (Nissim et al. 2011). We evaluated IAHD A^* on multiple sequence alignment (MSA) using the variant of HDA * in (Kobayashi et al. 2011), without the weighted- A^* preprocessing/upper-bounding step. The test set has 28 standard alignment problems for 5-9 sequences (HPC only).

Table 1 summarizes the results. We only consider problems which required ≥ 2 iterations on that cluster. In the discrete model, we assume the industry standard 1 hour granularity. In all our test problems, $E \leq 1$ and thus discrete $R^* = R^+$. The “# solved with zero overhead” column shows the number of problems where the discrete $R^* = 1$. The mean discrete R^* for all problems, on all 3 clusters, is significantly less than the theoretical worst case bound (4.0) and average case bound (2.67) for the doubling strategy. The continuous R^+ was never higher than 3.34.

References

- Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proceedings of ICAPS-07*, 176–183.
- Kishimoto, A.; Fukunaga, A.; and Botea, A. 2009. Scalable, Parallel Best-First Search for Optimal Sequential Planning. In *Proceedings of ICAPS-09*, 201–208.
- Kishimoto, A.; Fukunaga, A.; and Botea, A. 2012. Evaluation of a simple, scalable, parallel best-first search strategy. *arXiv:1201.3204v1*. <http://arxiv.org/abs/1201.3204>.
- Kobayashi, Y.; Kishimoto, A.; and Watanabe, O. 2011. Evaluation of Hash Distributed A^* in Optimal Sequence Alignment. In *Proceedings of IJCAI-11*, 584–590.
- Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proceedings of IJCAI*, 1983–1990.