Report Number: WUCSE-2010-29

2010

Cloud Computing for Scalable Planning by Stochastic Search

Authors: Qiang Lu, You Xu, Ruoyun Huang, and Yixin Chen

Graph search has been employed by many AI techniques and applications. A natural way to improve the efficiency of search is to utilize ad- vanced, more powerful computing platforms. However, expensive computing infrastructures, such as supercomputers and large-scale clusters, are traditionally available to only a limited number of projects and researchers. As a results, most AI applications, with access to only commodity com- puters and clusters, cannot benefit from the efficiency improvements of high-performance parallel search algorithms. Cloud computing provides an attractive, highly accessible alternative to other traditional high- performance computing platforms. In this paper, we first show that the run-time of our stochastic search algorithm in planning is a heavy-tailed distribution, which has a remarkable variability. Second, we propose an algorithm framework that takes advantage of cloud computing.
... **Read complete abstract on page 2.**

Cloud Computing for Scalable Planning by Stochastic Search

Complete Abstract:

Graph search has been employed by many AI techniques and applications. A natural way to improve the efficiency of search is to utilize ad- vanced, more powerful computing platforms. However, expensive computing infrastructures, such as supercomputers and large-scale clusters, are traditionally available to only a limited number of projects and researchers. As a results, most AI applications, with access to only commodity com- puters and clusters, cannot benefit from the efficiency improvements of high-performance parallel search algorithms. Cloud computing provides an attractive, highly accessible alternative to other traditional high- performance computing platforms. In this paper, we first show that the run-time of our stochastic search algorithm in planning is a heavy-tailed distribution, which has a remarkable variability. Second, we propose an algorithm framework that takes advantage of cloud computing.

2010-29

Cloud Computing for Scalable Planning by Stochastic Search

Authors: Qiang Lu and You Xu and Ruoyun Huang and Yixin Chen

Abstract: Graph search has been employed by many AI techniques and applications. A natural way to improve the e□ciency of search is to utilize advanced, more powerful computing platforms. However, expensive computing infrastructures, such as supercomputers and large-scale clusters, are traditionally available to only a limited number of projects and researchers. As a results, most AI applications, with access to only commodity computers and clusters, cannot benefit from the e□ciency improvements of high-performance parallel search algorithms. Cloud computing provides an attractive, highly accessible alternative to other traditional high-performance computing platforms. In this paper, we first show that the run-time of our stochastic search algorithm in planning is a heavy-tailed distribution, which

Type of Report: Other

ied and applied to several areas of automated planning, such as sampling possible trajectories in probabilistic planning (Bryce, Kambhampati, and Smith 2006) and robot motion planning (LaValle 2006). (Fern, Yoon, and Givan 2004) uses random walk exploration to lean domain-specific control knowledge.

This paper generally has two contributions. First, we show that the run-time distribution Monte Carlo Random Walk (MRW) algorithm in planning is a heavy-tailed distribution, which has a remarkable variability. Second, we propose a parallel MRW algorithm which takes advantage of short runs in thus heavy-tailed distribution. Our parallel MRW algorithm is a parallel stochastic search which use low frequency communication, even no communication between computing nodes which is perfectly suitable for cloud computing architecture.

The remainder of this paper is organized as follows: Section briefly reviews the cloud computing, SAS+ formalism of classic planning and Monte Carlo Random Walk method. Section explains details of MRW algorithm and its heavy-tailed run-time distribution. Section introduces basic parallel MRW algorithm and a communication technique to improve the efficiency and probability of solving problems. Section discusses the performance on standard planning benchmarks from the IPC-4 competition. Section contains concluding remarks and some potential directions for future work.

Background

Cloud computing

SAS+ Formalism

In this paper, we work on the SAS+ formalism (Jonsson and Bäckström 1998) of classical planning. In the following, we review this formalism and introduce our notations.

Definition 1. *A **SAS+ planning task** Π is defined as a tuple $\{X, \mathcal{O}, S, s_I, s_G\}$.*

- *$X = \{x_1, \cdots, x_N\}$ is a set of multi-valued* state variables*, each with an associated finite domain $Dom(x_i)$.*
- *$\mathcal{O}$ is a set of actions and each action $o \in \mathcal{O}$ is a tuple $(pre(o), \mathit{eff}(o))$, where both $pre(o)$ and $\mathit{eff}(o)$ define some partial assignments of variables in the form $x_i = v_i, v_i \in Dom(x_i)$. s_G is a partial assignment that defines the goal.*
- *S is the set of states. A **state** $s \in S$ is a full assignment to all the state variables. $s_I \in S$ is the initial state. A state s is a goal state if $s_G \subseteq s$.*

For a given state s and an action o, when all variable assignments in $pre(o)$ are met in state s, action o is *applicable* at state s. After applying o to s, the state variable assignment will be changed to a new state s' according to $\mathit{eff}(o)$: the state variables that appear in $\mathit{eff}(o)$ will be changed to the assignments in $\mathit{eff}(o)$ while other state variables remain the same. We denote the resulting state of applying an applicable action o to s as $s' = apply(s, o)$. $apply(s, o)$ is undefined if o is not applicable at S. The planning task is to find a **plan**, a sequence of actions that transits the initial state s_I to a goal state that includes s_G.

An important structure for a given SAS+ task is the domain transition graph defined as follows.

Definition 2. *For a SAS+ planning task, each state variable $x_i, i = 1, \cdots, N$ corresponds to a **domain transition graph (DTG)** G_i, a directed graph with a vertex set $V(G_i) = Dom(x_i) \cup v_0$, where v_0 is a special vertex, and an edge set $E(G_i)$ determined by the following.*

- *If there is an action o such that $(x_i = v_i) \in pre(o)$ and $(x_i = v_i') \in \mathit{eff}(o)$, then (v_i, v_i') belongs to $E(G_i)$ and we say that o is **associated** with the edge $e_i = (v_i, v_i')$ (denoted as $o \vdash e_i$). It is conventional to call the edges in DTGs as **transitions**.*
- *If there is an action o such that $(x_i = v_i') \in \mathit{eff}(o)$ and no assignment to x_i is in $pre(o)$, then (v_0, v_i') belongs to $E(G_i)$ and we say that o is **associated** with the transition $e_i = (v_0, v_i')$ (denoted as $o \vdash e_i$).*

Intuitively, a SAS+ task can be decomposed into multiple objects, each corresponding to one DTG, which models the transitions of the possible values of that object.

Monte-Carlo Random Walk

In Monte-Carlo Random Walk planning (Nakhost and Mller 2009), fast Monte-Carlo random walks are used for exploring the neighborhood of a search state. A relatively large set of states S in the neighborhood of the current state s_0 is sampled before greedily selecting a most promising next state $s \in S$. For example, a new random walk starts from s_0, builds a sequence of actions $o_0 \rightarrow o_1 \rightarrow ... \rightarrow o_k$ and changes s_0 to s. At the end of the random walk, s is evaluated by a heuristic function h, for instance by the FF heuristic, and added to S. When a stopping criterion is satisfied, the algorithm chooses a state in S with the minimum h-value to replace s_0.

The MRW method uniformly deals with both problems of local search methods: it quickly escapes from local minima and can recover from areas where the evaluation is poor. The MRW method does not rely on any assumptions about the local properties of the search space or heuristic function.

Monte-Carlo Random Walk Search

Alogorithm 1 shows the framework of Monte-Carlo Random Walk method. Given a SAS+ planning problem Π, MRW search builds a chain of states $s_I \rightarrow s_1 \rightarrow ... \rightarrow s_n$ such that s_I is the initial state, s_n is a goal state, and each transition $s_i \rightarrow s_{i+1}$ uses an action sequence found by RandomWalk exploring the neighborhood of s_i (Line 9). MRW search fails to find a solution when the minimum obtained h-value does not improve within MAX_STEPS times, or s_i is a dead-end

Algorithm 1: MRW(Π)

Input: SAS+ planning problem Π
Output: a solution plan

```
1  s ← s_I ;
2  plan ← ∅;
3  h_min ← h(s_I) ;
4  counter ← 0 ;
5  while s does not satisfy s_G do
6      if counter > MAX_STEPS or DeadEnd(s) then
7          s ← s_I ;
8          counter ← 0 ;
9      plan, s ← RandomWalk(s, Π) ;
10     if h(s) < h_min then
11         h_min ← h(s);
12         counter ← 0;
13     else
14         counter ← counter + 1;
15 return plan;
```

state (Line 6). In this case the MRW search simply restarts from s_I (Line 7). The algorithm return a solution plan which contains a sequence of actions changing state from s_I to a goal state s (Line 15).

RandomWalk procedure has three variations (Nakhost and Mller 2009). The base procedure uses pure random walk, where all applicable actions are equally likely to be explored. The other two procedures, MDA and MHA, use statistics from earlier random walks to bias the random action selection. The MDA and MHA enhancements can address the problems of MRW planning with high density of dead-end states and large average branching factors.

We study the runtime distribution of MRW procedure in different planning domains. Such random procedures often exhibit a remarkable variability in the time required to solve any problem instance. In our experiments, we run MRW solver "hanging" on a given instance hundreds of times with different random seeds. The runtime distribution of MRW procedure has an intriguing property: they are often characterized by very long tails or "heavy tails". See Figure 1.

Heavy-tailed distributions were first introduced by Vilfredo Pareto in the context of income distribution. It has been extensively studied and used to model plenty of real world phenomenas (Mandelbrot 1960)(Adler, Feldman, and Taqqu 1998)(Carla P. Gomes and Crato 1997) (Rish and Frost 1997)(Gomes, *et al.* 2000).

Parallel MRW Planning

Based on the study of heavy-tailed distributions, we present a parallel MRW procedure expecting to take advantage of short runs and significantly reduce solving time. Algorithm 2 shows the framework of parallel MRW procedure (PMRW). It simply use N processes

Algorithm 2: PMRW(Π)

Input: SAS+ planning problem Π
Output: a solution plan

```
1  for each processor P_i, 1 ≤ i ≤ N do
2      plan ← MRW(Π);
3      if plan is a solution then
4          Abort all other processors;
5  return plan;
```

to run MRW procedure independently (Line 2). The procedure will abort all other processes when a process find a solution (Line 4). Obviously, the solve time of the PMRW is the minimal solve time of N independent runs of MRW.

Suppose X and X' are the runtime variable of MRW and PMRW. We have

$$P(X' < x) = 1 - [1 - P(X < x)]^N \quad (1).$$

Let EX be the mean of X. Suppose $P(X < EX) = 0.2$ and $N = 8$, according to formula (1), $P(X' < EX) = 0.83$. Thus, even though the probability of short runs are low in MRW, the probability of hitting the same short runs are high enough to be accepted in PMRW. We can compute the expected mean of X' according to the runtime distribution of X. Table 1 gives the expected speedup(EX/EX') with different N of four instances.

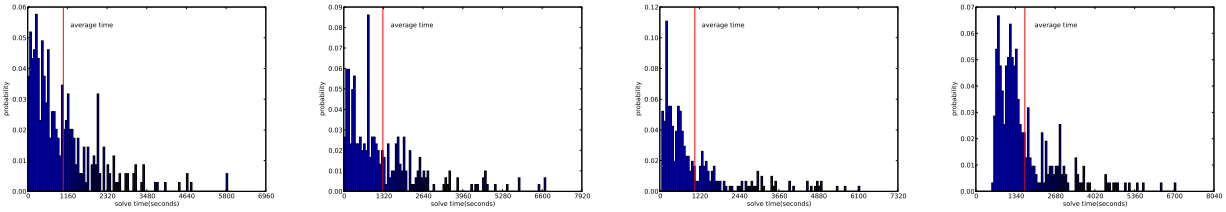
problem	expected speedup					
	2	4	8	16	32	64
air-17	4.76	7.64	11.40	15.73	19.83	22.92
tank-31	4.94	7.94	11.86	16.30	20.56	24.08
notan-45	4.76	7.64	11.40	15.73	19.83	22.92
sate-24	1.20	1.50	1.76	1.94	2.00	2.00

Table 1: Expected speedup of parallel MRW Algorithm.

Parallel MRW with Communication

Since the runtime distribution of MRW has large variability, the long/bad runs may cause our solver slower, even ending without finding any solution. Based on the simple Parallel MRW algorithm, we can increase the probability of hitting short/good runs by making processes communicate with each other in necessary condition. The aim of communication is replacing long/bad runs by good/short runs. We add a global priority queue Q indexed by heuristic value $h(s)$. Processes communicate with each other by writing message to or requiring message from Q. A message is defined as a tuple $(plan, s)$. *plan* is a action sequence which represents the search progress from initial state s_I to current state s.

We use MRW-C (Algorithm 3) to replace MRW procedure in PMRW algorithm. Algorithm 3 shows the



(a) airport-17 (b) tankage-31 (c) no-tankage-45 (d) satellite-24

Figure 1: The distribution of solving time in different instances.

Algorithm 3: MRW-C(Π, Q)

Input: SAS+ planning problem Π, Queue Q
Output: a solution plan

```
1  s ← s_I ;
2  plan ← ∅;
3  h_min ← h(s_I) ;
4  counter ← 0 ;
5  while s does not satisfy s_G do
6      if counter > MAX_STEPS or DeadEnd(s) then
7          plan, s ← ReadMessage(Q);
8          m_counter ← 0;
9          counter ← 0 ;
10     plan_temp, s_temp ← RandomWalk(s, Π) ;
11     if h(s_temp) < h_min then
12         if m_counter < MAX_M then
13             WriteMessage(Q, plan_temp, s_temp);
14             m_counter ← m_count + 1;
15         else
16             plan ← plan_temp;
17             s ← s_temp;
18             h_min ← h(s);
19             counter ← 0;
20     else
21         plan ← plan_temp;
22         s ← s_temp;
23         counter ← counter + 1;
24 return plan;
```

MRW procudre intergrading communication technique (MRW-C). Algorithm 3 has two changes compared to Q. First, when MRW stuck to a bad run, not getting any progress within MAX_STEPS walks or s_i is a dead-end state (Line 6), it will require a message from Q. If Q has messages, RequireMessage() will return a tuple $(plan, s)$ which is a possible good search condidate. Otherwise, it will return $(\emptyset, s_I)$. Second, when MRW finds a better state, it will write a message to Q (Line 13). The number of messages writed by a good run is at most MAX_M. This bound make sure that a good run will search forward after writing some good states.

There are two issues of communication technique in our experiments. One is when requiring message from Q, there are a lot of states having the same best heuristic value. Another is by replacing bad runs with good runs, some processes may search in the same local area of search space. It make some of them do repeat and no use work. We introduce a hamming distance H to address the above problems (Hamming 1950). The hamming distance between two states s_1 and s_2 is defined as:

$$H(s_1, s_2) = \sum_{\forall x_i \in X} d(s_1(x_i), s_2(x_i)).$$

$d(s_1(x_i), s_2(x_i))$ is the distance of $s_1(x_i)$ and $s_2(x_i)$ in x_i's domain transition graph G_i.

A big $H(s_1, s_2)$ means states s_1 is far away from s_2 in search space. Choose big H make processes search have high probability to search all good runs with less repeat work. When there are some states having the same best h value, we compute the hamming distance between these states and current bad state or dead-end state s. Then, we choose the state with the biggest H value and return it to MRW-C.

Experimental Results

Conclusions

References

Adler, R.; Feldman, R.; and Taqqu, M. 1998. *A Practical Guide to Heavy Tails*. Boston: Birkh[ddota]user.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Sequential monte carlo in probabilistic planning reachability heuristics. In *ICAPS*, 233–242.

Carla P. Gomes, B. S., and Crato, N. 1997. *Heavy-tailed distributions in combinatorial search*. Principles and Practice of Constraint Programming-CP97.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*, 191–199.

Gomes, *et al.*, C. P. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning* 24(1):67–100.

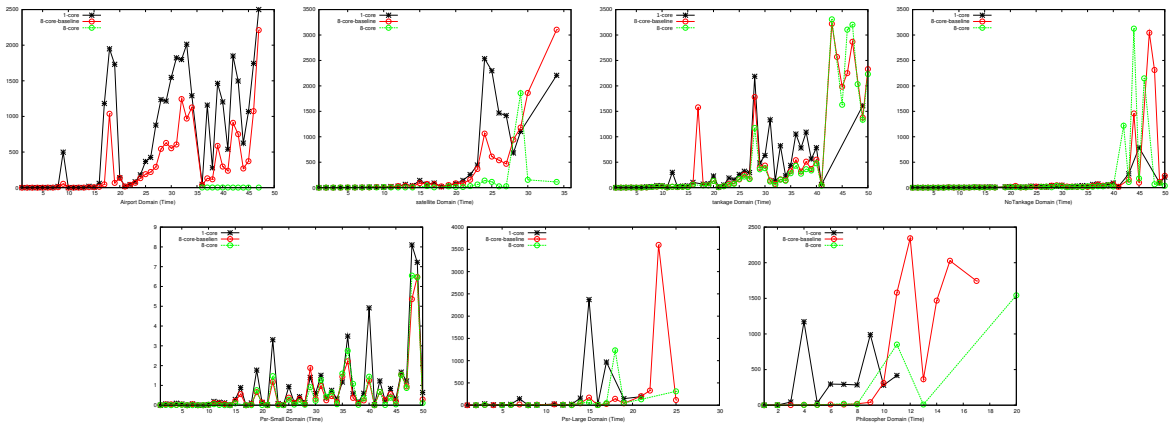


Figure 2: Solution time on IPC4 domains.

	1-core			8-core-base			8-core		
domain	#soln	avg	min	#soln	avg	min	#soln	avg	min
air36	8	46	36	8	37	35	8	43	30
air37	8	1159	131	8	152	101	8	155	38
air38	8	279	117	8	109	84	8	100	80
air39	5	1464	586	8	949	248	8	742	254
air40	6	1204	296	8	462	186	8	578	206
air41	8	538	237	8	195	130	8	190	143
air42	4	1848	910	6	1120	344	7	663	335
air43	3	1497	751	8	1028	412	8	420	138
air44	8	625	269	8	253	194	8	285	203
air45	6	1068	371	8	446	257	8	448	283
air46	2	1744	1074	5	1394	1006	5	1106	656
air47	2	2500	2212	7	1336	871	7	1502	111
tank26	8	322	188	8	255	204	8	218	145
tank28	3	2188	405	6	1782	482	3	1174	179
tank29	8	485	352	8	389	300	8	360	277
tank30	8	632	405	8	434	345	8	382	261
tank31	8	1333	110	8	141	31	8	120	20
tank33	8	823	213	8	163	78	8	157	71
tank34	8	232	150	8	144	107	8	135	105
tank35	8	437	219	8	324	266	8	283	201
tank36	5	1054	907	8	540	313	8	431	179
tank37	8	780	406	7	317	194	8	274	207
tank38	8	1091	499	8	511	260	8	363	251
tank39	8	562	424	8	366	257	8	337	283
tank40	8	785	469	8	550	388	8	478	307
tank45	-	-	-	7	1984	1005	7	1629	628
tank48	-	-	-	-	-	-	2	2033	1647
tank49	8	1612	1128	8	1376	1042	8	1334	920
tank50	-	-	-	7	2330	1901	7	2231	1900

Table 2: Cpmparison of 1-core, 8-core baseline and 8-core Monte Carlo Algorithms.

Hamming, R. W. 1950. Error detecting and error correcting codes. In *Bell System Technical Journal*, volume 26(2), 147–160.

Jonsson, P., and Bäckström, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence* 100(1-2):125–176.

LaValle, S. M. 2006. Planning algorithm. In *Cambridge University Press, Cambridge, U.K.*

Mandelbrot, B. 1960. The pareto-levy law and the distribution of income, internat. In *Econom*, volume 1, 79–106.

Nakhost, H., and Mller, M. 2009. Monte-carlo exploration for deterministic planning. In *IJCAI*, 1766–1771.

Rish, I., and Frost, D. 1997. *Statistical analysis of backtracking on inconsistent CSPs*. Principles and Practice of Constraint Programming-CP97.