DEIM Forum 2011 C3-5

クラウド環境での分散ゲーム木探索実現への取り組み

横山　大作† 　　喜連川　優†

† 東京大学生産技術研究所　〒 153–8505 東京都目黒区駒場 4-6-1
E-mail: †{yokoyama,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　クラウド環境においては従来、web アプリケーションのようなトランザクションスループット追求型のアプリケーションを目的とした研究開発が行われてきたが、探索問題などその枠組みに含まれないアプリケーションについても適用が望まれる。本研究では、大規模探索問題の一種であるゲーム木探索をターゲットとし、コンピュータ将棋プレイヤ「激指」に対してクラウド環境を想定した分散計算適用を試みた。

キーワード　クラウド、分散計算、ゲーム木探索問題

An Implementation of Distributed Game-Tree Search on Cloud

Daisaku YOKOYAMA† and Masaru KITSUREGAWA†

† Institute of Industrial Science, The University of Tokyo
Meguro-ku Komaba 4–6–1, Tokyo, Japan, 153–8505
E-mail: †{yokoyama,kitsure}@tkl.iis.u-tokyo.ac.jp

1. はじめに

1.1 クラウドコンピューティング

近年、計算機の低廉化と仮想化技術の進展に伴い、クラウドコンピューティングと呼ばれるオンデマンド大規模計算環境が急速に広まりつつある [1] [2]。この環境は、大規模なクラスタ計算機を仮想化技術によって部分的に割り当てながら使用することで、多数のユーザで共有しながら利用することを可能にしたものである。要求に対して即座に計算機を提供することが可能であり、利用したリソースに応じた分だけの課金がなされる “pay-as-you-go” のモデルである点に特徴がある。計算規模を拡大する際には、単体計算機の性能向上を図る「スケールアップ」ではなく、計算機の利用台数を増やす「スケールアウト」の手法が適用されるため、この手法によって性能を向上できるようなアプリケーションの構築法を用いなければならない。典型的な利用領域としては、web 通販に代表される e-commerce サイト [3] や、オバマ大統領就任前の意見集約サイト “Change.gov” やエコポイント登録サイトなどの web サービスサイト [4]、などが挙げられる。これらは、多数のユーザによって利用されるサービスアプリケーションであり、ユーザ一人一人へのサービス提供に必要な計算量は小さいが、非常に多くのユーザの処理が短期間に要求されるような利用形態である。単位時間あたりの処理個数を増やしていくことが望まれるスループットを求められるアプリケーションであり、スケールアウトによる性能向上が比較的容易であるためクラウドに適した分野であると言える。クラウド上のプログラミングフレームワークを提供している PaaS (Platform as a Service) と呼ばれる環境においても、プログラミング対象としては web アプリケーションを想定していることが多い [5]。

クラウドのもう一つの典型利用例として、データストレージとしての用途とその上でのデータインテンシブな計算アプリケーション実行が挙げられる。大量な文書、画像などの大容量データを、クラスタ上の安価な分散ファイルシステムに格納保存するサービスは広く提供されており、前述のクラスタ計算リソースから利用することで、メールサービスや画像投稿サイトなどのサービスを容易かつスケーラブルに実現可能になっている [6]。また、格納された大量データに対して解析、学習などの処理を行うデータインテンシブな計算プラットフォームとしての利用も試みられている。特に、データを key-value ペアとして格納し、そのデータ構造を利用してデータに対する操作適用をスケーラブルに実現する MapReduce [7] プログラミングモデルは、クラウドの資源特性によく適合するものとして広く利用されている。MapReduce を用いて機械学習のアルゴリズムを記述し、大規模データを用いた学習を可能にする研究 [8] なども行われており、データ処理アプリケーションのクラウド利用は急速に広まりつつある。

しかし、必ずしも全ての問題領域がクラウド環境に適用しやすいわけではない。MapReduce はごく単純な処理の内部依存関係を持つアプリケーションに対象を絞ったプログラミングフレームワークであるが、その適用が難しい問題も多い。探索問

題はその一つである。探索問題は、複雑な依存関係にある多数の計算から構成されており、MapReduce では記述が難しい。本論文では、探索問題の一種であるゲーム木探索を対象問題とし、予備的な実装実験を通してクラウド環境における分散計算の適用への知見を得ることを目指した。

1.2 ゲーム木探索のクラウド環境への適用

探索の分散化における対象アプリケーションとして、筆者らが開発を行っているコンピュータ将棋プレイヤ「激指」を用いることとした。激指は世界コンピュータ将棋選手権で過去 10 年間に 4 度優勝するなど、現時点でトップレベルの強さを持つ将棋プレイヤの 1 つである。

コンピュータ将棋プレイヤは、巨大なゲーム木のミニマックス値を求める大規模探索アプリケーションである。激指においては、2.5 倍の速度向上を達成すると相対的な強さとしては 2 勝 1 敗程度の勝率が得られることがわかっている。これは、ゲーム一般で相対的な強さの指標として用いられる「レーティング」に換算すると、100 点程度の向上に相当する。これはすなわち、強いプレイヤを実現するためには、指数関数的に計算量を増やさなければならないことを意味しており、現時点でのコンピュータ将棋プレイヤにおいて人間のトッププレイヤに迫る強さを実現するためには、大量の計算を必要とする。激指はマルチスレッドを用いた並列化が実装されており、共有メモリを持つ並列計算機上での高速探索が可能である。しかし、メモリが共有されていない PC クラスタなどの計算環境での分散計算はまだ実装されていない。探索の高性能化を図るためには、大規模分散環境の利用が必要な段階にある。

また、ゲーム木探索をクラウドコンピューティング環境上で行うためには、分散計算化に加えて、いくつかの課題を解決しなければならない。

第一に、クラウド環境においては、ほぼ無限の計算機リソースが利用可能であると考えてよく、また、計算機構成が容易に変更可能であるため、アプリケーションの実行システムはその特性を活かし、スケーラブルかつ容易に構成変更が可能であるようにすべきである。つまり、計算に参加する計算機を必要に応じて増やし、利用者の利便性を向上させられることが望まれる。

第二に、プログラムの分散化に必要な労力を低減したいという要求も当然ながら存在する。クラウド環境の魅力の一つに短い期間での開発・実行環境の整備という特質があるため、これを活かせるような短期間・高生産性の開発を行うことが望まれる。並列処理向きプログラミング言語は過去多数提案され、高性能な分散計算を比較的容易な記述で実現することが可能な分野も多く存在するが、激指のように、実問題に対する大規模なアプリケーションの並列化に際しては、プログラミング言語やプログラミングモデルを大きく変更することには労力を必要とし、開発コストが極めて大きくなってしまう。

本論文では、これらの課題をふまえ、激指の分散化を行い、その評価を通して現状の技術の課題を明かにすることを目指す。

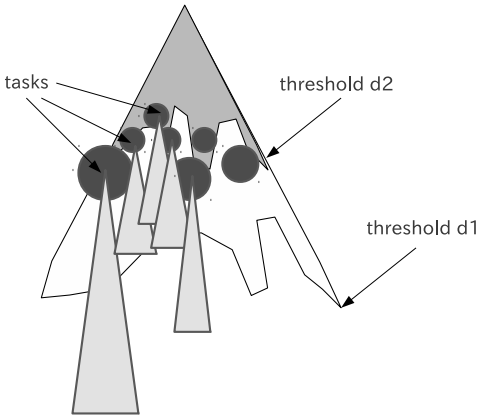

図 1　分散激指のタスク分割

2. 激指への分散計算の適用

2.1 設　　計

分散化に際しては、静的な探索木による単純なタスク分割を行うこととした。また、全体の制御構造としては、1 台のマスタが多数のワーカの制御を行う、単純なマスタワーカ方式をとることにした。

探索開始時、まずマスタ激指が、探索開始 (root) 局面において比較的短い探索を行い、root 近くの探索木を作成する。激指の場合、実現確率による探索打ち切り [9] を採用しているので、打ち切り閾値 d_1 を小さくして探索を行うことで、実現確率に従って、深く読むべき所を深く読むという探索深さ制御が行われる。この探索の時に、打ち切り閾値とは別の、より小さい閾値 d_2 を設定し、閾値 d_2 に達するまでの root 近傍のゲーム木については、その構造をメモリ上に記憶する。図 1 に閾値 d_1 と d_2 の探索木の様子を示す。これによって、閾値 d_1 程度の信頼性がある閾値 d_2 以下の探索木が作成される。

得られた探索木の葉局面を全て独立したタスクと見なし、その各々の葉局面から $\alpha\beta$ 探索の探索窓を $(-\infty, \infty)$ と設定し、ワーカ激指により探索を行う。他の局面の探索結果を利用せず、タスク間に依存関係がないため、全ての葉局面のタスクは並列に実行することができる。葉局面の探索打ち切り閾値は、探索全体の目標打ち切り閾値 d から、root から葉局面に至るまでの実現確率指標を引いたものとし、全体として打ち切り閾値 d に相当する深さまで探索できることを目指す。

全てのタスクの実行を終えると、全ての葉局面についての評価値と応答手順が得られるため、それを用いてマスタ激指が root までの min-max 木を作成し、探索全体としての評価値と最善応答手順を得る。

2.2 分散計算処理系

本論文では、激指に対して GXP make [10] による分散計算の適用を試みた。GXP make は田浦らが提案、実装している分散計算プラットフォームである。通常の Makefile のシンタックスによりタスク間の依存関係を記述しておくと、実行可能になったタスクのうち依存関係のない部分を同時にスケジューリングし、並列実行を行うことができる。タスク間のデータ受け

渡しはファイル経由で行われ、ユーザは通信機構など分散計算のためにプログラムを変更する必要がない。

GXP make は大規模なワークフローと呼ばれるタスクグラフの並列処理を目的として開発されているが、激指のように短時間での応答性が要求されるアプリケーションについては、十分な性能が得られるか、充分検証されてはいない。このようなアプリケーションについて適用を試みることで、ワークフロー処理系の適用範囲を広げるとともに、性能や機能に関する要求が明かになると期待できる。

2.3 実　　装

a) タスク生成とスケジューリング

GXP make による分散計算は、タスクをファイル経由で受け渡す。本実装で、全ての葉局面からの探索を実行する際には、葉局面の盤面配置情報と探索閾値を記入した `XXX.prob` というファイルを葉局面数だけ作成し、図 2 の makefile を指定して GXP make を呼ぶ。

図 2 は、make コマンドのシンタックスによるタスクの依存関係指定ファイルである。`PROB` ディレクトリ中の全ての `XXX.prob` ファイルに対し、`python worker.py` 以下のコマンドを実行することで、`XXX.result` を作成する。

GXP make は、計算に参加しているノードに `XXX.prob` タスクを割り当て、それぞれのノード上で `python worker.py` コマンドを実行する。このコマンドにより、激指が `XXX.prob` ファイルを読み込み、指定された葉局面と探索閾値で探索を行い、結果が標準出力経由で `XXX.result` ファイルに出力される。

1 個のタスクが終了し、計算ノードが暇になると、GXP make のスケジューラが自動的に残っているものから 1 つタスクを割り当てる。このように、動的なスケジューリング機能があるため、必要実行時間の異なるタスクが混在している今回の状況でも、効率よく分散計算を行うことができる。

このような制御を実現するために、激指に加えた変更点は

- 浅い探索による探索木生成と、メモリ上での保持
- 局面と探索条件、および探索結果のファイルへの書き出し、読み込み
- メモリ上の探索木による min-max 値の計算

のみである。タスクデータの通信に関しては NFS(共有ファイルシステム) に、タスクの実行制御については GXP make に、それぞれ依存することで、このような簡単な変更で分散探索が実現できた。

タスク分割の際の粒度は打ち切り閾値を設定することで自由に決められるが、本論文の実験では、300〜10000 個程度のタスクが生成されるような設定を用いた。

b) ワーカ起動時間の短縮

GXP make は、`XXX.prob` を 1 つのタスクとして、記述されたコマンドをタスク毎に実行する。激指の起動には、メモリの初期化等の処理のために数秒程度の時間が必要となるが、葉局面のタスクは 1 秒以下の探索時間しか要しないものも多数含まれるため、タスク毎にワーカ激指を起動し、葉局面を読み込んでいると、激指の起動時間が無視できない長さとなり並列実行で速度向上が得られなくなる。

これを防ぐために、ワーカ激指は各計算ノードに 1 つずつあらかじめ起動しておき、局面が入力されたら即座に探索を行い結果を返す、という手順を繰り返せるような実装とした。GXP make からはラッパプログラム `worker.py` が呼ばれ、これがその計算ノードで立ち上がっているワーカ激指とソケット経由で通信する。

2.4 使用リソース量の調整

ゲーム木探索において、異なる root 局面の探索木はその複雑さが大きく異なることがしばしば観察される。このため、探索の制御法にもよるが、対局中の局面毎に必要な思考時間が大きく変化してしまう。激指で採用しているのは、実現確率がある閾値に達するまで探索を行うという制御方法であり、有力な手が少ない序盤や、「この一手」以外は大きく形勢を損ねるしかない一本道の読みが多い終盤においては探索時間が短く、反対に有力な展開が多数存在するような中盤においては探索時間が長くなりがちである。

一方、クラウド環境においては、リソースを任意の時点で必要なだけ割り当てることができるという大きな利点が存在する。この利点を利用し、複雑な局面ではリソースを増やして探索を行い、探索時間の変動を少なくして対局者の満足度を上げること試みた。

前述のように、本システムの分散探索においては、必要な root 近傍の探索木を作るため浅い読みを行う。この浅い探索木の複雑さは、探索木全体の複雑さと相関があると考えられるため、この浅い木の探索時間をもとに分散探索に用いるリソース量を決定することとした。

予備実験として、実験環境上で、同一局面で浅い読みと深い読みを行ったときの探索時間の関係を求めた。探索対象としたのは実戦棋譜からランダムにサンプリングした 100 局面 (サンプル A) である。浅い読み (探索深さに相当するパラメタを 10 に設定したとき) の探索時間と、深い読み (深さ 14 と 16 に設定した場合) の探索時間を測定した。結果を図 3 に示す。局面毎にばらつきが大きいが、相関があることは見て取れる。実線は最小二乗法により得られた、深さ 14 と 16 の探索時間予測直線である。今回は、極めて単純ではあるが、この予測直線をもとに探索時間を予測し、その時間がある閾値内にとどまるように探索リソース量を決定することとした。

なお、参加計算機の数を変更する際には、最初から最大構成のリソースを確保しておき、その一部を利用しないことで構成変更をシミュレートしている。制御用に利用しているセッションは接続し続けており、新規リソース確保に伴うオーバヘッドは反映されていない。

3. 評　　価

3.1 GXP make による分散化

今回の実装において、探索が高速化されるかを簡単な実験で確認した。PC クラスタを用い、計算ノード数を 60 として実験した場合、同一局面での探索においてノード数 1 と比較してほぼ 60 倍の速度向上が得られた。すなわち、分散探索自体の実装オーバヘッドは充分少なく、効率よく探索できていると言

```
PROB_DIR=prob
ALL_TARGETS=$(patsubst $(PROB_DIR)/%.prob, $(PROB_DIR)/%.result, \
        $(wildcard $(PROB_DIR)/*.prob))

all: $(ALL_TARGETS)

$(ALL_TARGETS) : $(PROB_DIR)/%.result : $(PROB_DIR)/%.prob
        python worker.py $< | tee $@ | \
        xargs echo $(shell echo $< | sed -e 's/.*dg\([0-9]*\).prob/\1/')
```

図 2　Makefile による激指の分散探索

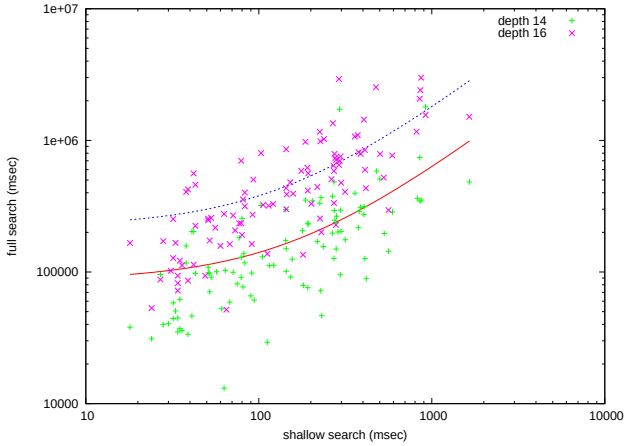

図 3　浅い読みによる問題サイズ予測

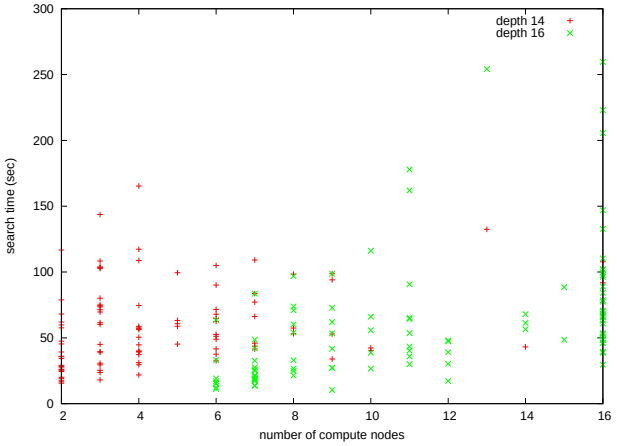

図 4　問題サイズ予測に基づく並列度制御

える。

ただし、分散化を行わない元の激指と比較すると、ほとんど速度向上が得られていない。予備的な評価では、分散化を行うと 10 倍以上速度が低下してしまい、60 ノード程度ではその速度低下を埋め合わせるに至らないためである。これは、ワーカのタスクが $\alpha\beta$ ウィンドウを利用せず、十分な枝刈りができないこと、transposition table の共有が行われないため、重複した計算を無駄に行ってしまうこと、などの原因によると考えられる。また、実現確率打ち切りの再探索アルゴリズムも激指本来のものとは異なっており、逐次探索と探索木が異なるため、対局における強さも異なると予想される。

再探索アルゴリズムをより忠実に再現しようとしたり、枝刈りによる動的なタスクの変更を実現しようとしたりすると、現在の GXP make を利用した処理では制御が不自由な部分が多い。make は基本的に処理開始時に静的にタスクグラフを作成してスケジューリングを行うため、動的なタスク変更を実現することが難しくなる。処理系に、キューイングされたタスクを適切にスケジューリングするような機能を追加することで、このようなタスクへの対応が可能ではないかと考えている。

3.2　問題サイズ予測に基づく並列度制御

図 3 で得られた探索時間予測モデルを元に、1 つ 1 つの局面での探索時間を一定時間以内に抑えたままで、なるべく少ない計算リソースで探索を実行できるよう、分散探索の並列度を制御することを試みた。

クラウドを模した環境として、PC クラスタにクラウド管理ソフトウェア Eucalyptus [11] を導入し、実験環境とした。利用した構成は以下の通りである。

- Hardware: Xeon E5530 2.40GHz, 8 cores, 24GB memory, Gbit ether
- dom0: xen 3.2.1, Linux 2.6.26
- domU: 1 core, 2GB memory, 20GB localdisk
- domU: xen 3.2.1, Linux 2.6.24

ここで、dom0 は VM を動かすプラットフォームとなるホスト OS、domU は VM によるゲスト OS である。

浅い読みの探索時間をもとに、深い読みの探索時間を予測し、その探索時間が 40 秒となるように並列度を決定した。ここで、探索時間は並列度に対して理想的に減少し、並列オーバヘッドは発生しないと仮定した。実戦棋譜から新たに 100 局面のランダムサンプリングを行い (サンプル B)、この局面での並列探索時間を測定した。結果を図 4 に示す。横軸はそれぞれの局面で必要であると求められた並列度、縦軸はその並列度で探索した際の探索時間である。

おおむね 100 秒以下に押さえられていると言えるが、ばらつきが大きく、100 秒を越える時間が必要だった局面も少なくない。なお、深さ 16 の一部の問題で探索時間が伸びているものがあるが、これは本実験の最大リソースを 16 に制限したため、問題サイズに対して利用可能な計算リソースが不足していたためである。

また、100 局面の探索に必要とした総 CPU 時間を図 5 に示す。ここで、総 CPU 時間とは、探索時間に利用した CPU コ

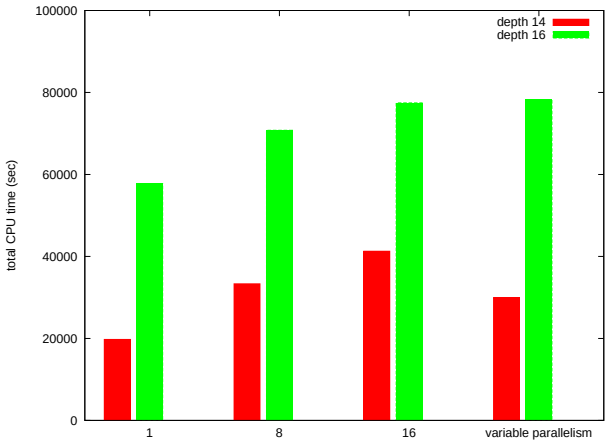

図 5　総 CPU 時間

ア数をかけたものとする。並列度を増やした場合、並列オーバヘッドが存在するため、同一局面の探索に対して利用した総 CPU 時間は増加してしまう。横軸の分類は順に、全ての問題を並列度 1、並列度 8、並列度 16 に固定して探索した場合と、問題サイズ予測に基づいて並列度制御を行った場合である。

深さ 14 の場合、並列度 8 に固定した場合より少ない総 CPU 時間で探索が実現されている。これはすなわち、利用者の許容する範囲を守った状態で、無駄な CPU リソース利用を削減できたことになる。

また、深さ 16 の場合は、並列度 16 以上の総 CPU 時間がかかってしまっている。総 CPU 時間は、実験で用いた問題のうち規模の大きな一部のものによって大きく影響を受けるため、そのような問題における外乱などが原因として考えられるが、詳細な理由についてはさらに検討が必要である。

4. 関 連 研 究

ゲーム木の分散探索は多数の方式が提案されており [12] [13] [14]、今回用いた単純な手法と比較して効率も高いが、使用するリソース量を変動させることは考えられていない。

将棋を対象とした研究として、GPS 将棋は、300 ノード以上の大規模分散環境での探索を実現した [15]。彼らは、分散実装において比較的単純な問題分割構造を利用し、探索効率はある程度低下するとしても、単純に計算機数を増やすことで全体の計算速度を向上させることを目指す、という本稿と同様のアプローチをとった。GPS 将棋の手法では探索木に対して計算リソースを静的に割り当てているため、リソース量を変化させることは難しい。

5. お わ り に

本論文では、コンピュータ将棋プレイヤ「激指」を題材とし、クラウド環境を意識した分散計算を適用することを試みた。ワークフロー処理系である GXP make を利用することで、少ない手間で分散計算化が実現でき、十分な性能で動作できることが確認された。しかし、分散化アルゴリズムが単純なため、ゲームプレイヤの強さとしてはそれほど良い結果は得られなかった。

また、単純な戦略により計算リソースの量を調整し、大量の計算が必要と予想されるときにリソース量を増やすことで、利用者の利便性の向上と計算機の利用効率を向上させることを試みた。予備的な実験を行った結果、ある程度探索時間の変動を押さえることができたが、ばらつきが大きく、より適切なリソース管理を行うことが必要であると考えられる。

謝　　辞

本研究の一部は総務省委託研究「セキュアクラウドネットワーキング技術の研究開発 (クラウドサービス連携技術)」において実施された。

文　　献

[1] Aaron Weiss. Computing in the clouds. *netWorker*, Vol. 11, pp. 16–25, December 2007.

[2] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, pp. 5–13, 2008.

[3] Amazon. Amazon elastic compute cloud. http://aws.amazon.com/jp/ec2/.

[4] salesforce. Force.com cloud platform. http://www.salesforce.com/platform/.

[5] Google. Google app engine. http://code.google.com/intl/ja/appengine/.

[6] Amazon. Amazon simple storage service. http://aws.amazon.com/jp/s3/.

[7] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, December 2004.

[8] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *Proceedings of Neural Information Processing Systems Conference (NIPS)*, pp. 281–288, 2006.

[9] Yoshimasa Tsuruoka, Daisaku Yokoyama, and Takashi Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, Vol. 25, No. 3, pp. 145–152, 2002.

[10] Kenjiro Taura. Gxp : An interactive shell for the grid environment. In *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp. 59–67, January 2004.

[11] Eucalyptus. http://www.eucalyptus.com/.

[12] Rainer Feldmann, Peter Mysliwietz, and Burkhard Monien. Distributed game tree search on a massively parallel system. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, pp. 270–288, London, UK, 1992. Springer-Verlag.

[13] Mark G. Brockington and Jonathan Schaeffer. APHID: Asynchronous parallel game-tree search. *Journal of Parallel and Distributed Computing*, Vol. 60, No. 2, pp. 247–273, 2000.

[14] John W. Romein, Aske Plaat, Henri E. Bal, and Jonathan Schaeffer. Transposition table driven work scheduling in distributed search. In *AAAI/IAAI*, pp. 725–731, 1999.

[15] 金子知適, 田中哲朗. 最善手の予測に基づくゲーム木探索の分散並列実行. 第 15 回ゲームプログラミングワークショップ, pp. 126–133, 2010.