クラウドアプリケーションとしてのコンピュータ将棋プレイヤ「激指」

横　山　大　作[†1]

コンピュータゲームプレイヤ「激指」において、大規模なゲーム木を高速に探索するために、近年広く利用され始めているクラウドコンピューティング環境を意識した分散計算の適用を試みた。分散計算処理系 GXP make の利用によりプログラムの変更を最小限にとどめることができ、分散実行のオーバヘッドも充分小さいものであることが確かめられた。しかし、探索全体の強さに関しては良い結果が得られず、分散アルゴリズムの改良の必要など課題が明らかになった。

1. はじめに

近年、計算機の低廉化と仮想化技術の進展に伴い、大量の計算機をユーザ間で共有し、需要に応じて仮想計算機をユーザに割り当てて利用する、クラウドコンピューティングと呼ばれる計算環境が普及を始めている。クラウドコンピューティングは、

- 物理的に計算機を持つ必要がなく、設置スペース、電源、空調などの準備が必要ない。また、ハードウェア故障への対応などの管理コストも発生しない。
- 数百台、数千台という計算機を同時に利用可能であり、実用上はほぼ制限のない、仮想的に無限の大きさの計算資源が利用可能である。
- 計算資源は必要な時点で即座に確保、利用することが可能である。人間が関与する利用申請手続きや、物理的計算機導入に必要な時間が存在しない。
- 課金においては、利用した資源量の分だけを支払えば良い。利用開始時の初期投資を必要とせず、高額な固定費も存在しない。

という特長を持ち、ウェブアプリケーションなどの提供基盤として広く利用され始めている。

アプリケーションサービスを提供する側にとって、クラウド環境は、試験サービス期間やサービス開始時のような小規模運用において低額で利用することができるとともに、ユーザ拡大時には大きな改造を加えることなくそのまま規模を拡大できる、「スケールアウト」が行いやすい環境として、利便性が高い。また、利用ユーザ数が時間によって大きく変動するようなアプリケーションにおいては、利用計算機構成を需要に応じて変化させることで、必要以上のコストを払うことなくサービスを提供することができる。

このような利点から、クラウドコンピューティングはいくつかの分野で広まり始めている。現在よく利用されている分野としては、

- ストレージサービス (Evernote, dropbox など)
- メールサービス、ショートメッセージ、SNS (Gmail, twitter など)
- EC サイト
- 企業内業務システム

などが挙げられる。これらの分野は、1 つのサービスを多数のユーザによって利用するアプリケーションであり、クラウドはそのようなワークロードをさばく実行環境としての利用に適した特性を持つ。しかし、High Performance Computing(HPC) など、1 つの巨大な計算を実行する分散計算環境としての利用も始まりつつある。代表的なクラウド計算環境である Amazon EC2[1] は、高速な CPU と大量のメモリ、高速ネットワーク帯域を備えた HPC 用のクラウドを提供し始めた。これは、これまで計算環境として利用されてきた PC クラスタと比較して、遜色ない性能と使い勝手を備えており、今後はこのような分野においても利用が広まると期待される。

このように、計算を行うための基盤システムとしてのクラウドはすでに広く提供されており、利用も活発である。しかし、クラウド上のプログラミング環境についてはまだ発展途上であるといえる。クラウド環境を意識した分散プログラミング環境としては、MapReduce[2]

†1 東京大学
The University of Tokyo

の処理系である Hadoop や、Google App Engine[3]、Dryad[4] などが提案されており、利用可能になっている。これらはタスクの容易な記述と分散実行を可能にし、高いスケーラビリティ、動的負荷分散、耐故障などの機能を実現している。しかし、これらは限定された制御構造に特化したプログラミングモデルを持ち、問題によっては必ずしも適用が容易とは言えない。また、既存のプログラムを分散化する際にはそれなりに大きな労力が必要となる。

本論文では、クラウドコンピューティング環境を用いた分散計算を探索問題の一種であるゲーム木探索に適用し、容易に巨大な計算を行うことを可能にする試みを紹介する。ここでは、クラウドの実行環境を利用した分散計算を実現するのみならず、既存のアプリケーションをいかに簡単に分散化するか、という面にも注目して実装を試みた。

2. コンピュータ将棋プレイヤの分散化

本論文では、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」を対象アプリケーションとし、分散計算の適用を試みた。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。

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。

激指の分散化において考慮すべき課題点を、コンピュータ将棋プレイヤというアプリケーションが持つ性質と、クラウドという環境が持つ性質とに分けて、以下に整理する。

2.1 コンピュータ将棋プレイヤ

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の指標であるレーティングが 100 点程度向上することが知られており、強いプレイヤを実現するためには大量の計算を必要とする。

ゲーム木は実際には木構造ではなく、同一の局面に至るグラフ構造を持っていることが多いため、その同一局面の計算結果を記憶・共有し、再計算を防ぐ必要がある。この、transposition table 機構による重複排除が利用できないと、分散計算を行っても無駄な計算ばかりで探索速度が向上しない。また、実用的なプレイヤにおいては、枝刈りなどを用いて探索効率を上げるために様々な工夫を施しているため、探索途中の子問題間に複雑な依存関係が発生し、並列化が難しくなることが多い。共有メモリを用いた並列計算においてはまだ対応が可能ではあるが、よりレイテンシが大きい分散計算においてはこれらの問題が顕在化する。そのため、ゲーム木の並列・分散計算は以前より多数研究されてきたが、大規模な分散計算を適用することは困難を伴うものであった。

また、ゲームプレイヤは一般的に、対局相手である人間が許容できる時間制限での求解が望まれるため、数秒から数分程度の比較的短い時間で探索を終了することを要求されることが多い。従来の大規模計算では、より長い時間制約での計算をターゲットとしていることが多いため、計算を開始するまでのレイテンシが長いなど、既存の技術をそのまま適用した場合には要求を満たしにくい。

これらの問題点に加えて、大規模分散計算における一般的な問題点、

- 負荷分散
- 耐故障性の実現

についてももちろん考慮する必要がある。特に負荷分散については、探索問題は一般的に部分問題の計算量が予測しにくく、負荷の不均衡が生じがちであるため、動的負荷分散などの手法による対応は必須であるといえる。

2.2 クラウド環境における分散化

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いため、分散計算システムのスケーラビリティが重要要素となる。すなわち、前述のようなゲーム木探索アプリケーションから要請される特性を実現する際、計算に参加するノード数が増加した際にボトルネックを生じないような実装を行う必要がある。

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もそれに伴って容易に構成変更が可能であることが望まれる。つまり、システムのインストール、計算機構成変更に伴う設定変更、アプリケーション起動などの過程が短時間で行えなければならない。

さらに、アプリケーションに独立した話として、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が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。

2.3 方　　針

本論文においては、大規模探索を分散化する際に考慮すべきこれらの課題について考慮した結果、激指に対して GXP make[5] 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。実行環境についても、実行に必要なのはローカルなプロセス起動機構と共有ファイルシステムのみであり、あらかじめ計算ノードにライブラリをインストールするなどの準備が必要ないため、クラウド環境のように新たな計算機が大量に利用可能になる場合においても、速やかに実行環境を整えることができる。また、依存関係解析により結果に必要なタスクのみを実行することができ、タスクの中間生成物がファイル上に残っていることと合わせて、障害発生時にも必要最低限のタスクのみを再実行することで結果が得られる、自然な耐故障性を実現している。

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。本論文では、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を明かにすることを目標とする。

3. 激指の分散化

3.1 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[6] 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。

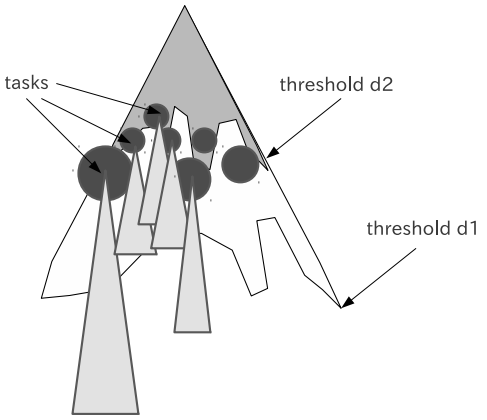

図 1　分散激指のタスク分割

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。

3.2 実　　装

並列にタスクを実行するための処理系については、GXP make を用いた。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

```
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

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。

3.3 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。

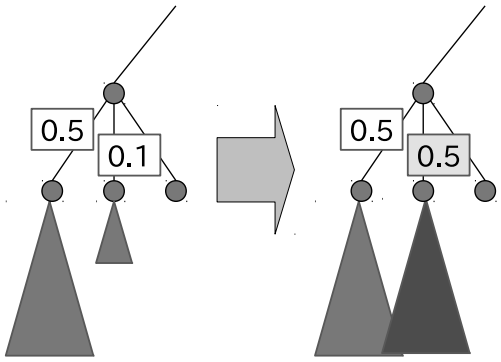

図 3 有望手の再探索

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。

3.4 実現確率打ち切り方式における再探索への対応

実現確率打ち切りアルゴリズムには、有用な局面の再探索という重要な手順が存在する。

今、**図 3** の左側のように、ある局面に対して子局面の実現確率が 0.5、0.1 のように定まったとすると、実現確率の大きい (より起きやすい展開の)0.5 の子局面はより深くまで探索し、0.1 の局面は浅くしか探索しないことになる。ここで、実現確率 0.1 の子局面が最善の評価値を返してきた時、浅い読みの結果が min-max 探索の結果として採用されることになり、探索全体の信頼性が落ちることになる。これを防ぐために、最善の評価値を返した子局面に関しては、最善手である場合に相当する大きな実現確率を与え、再度探索をし直

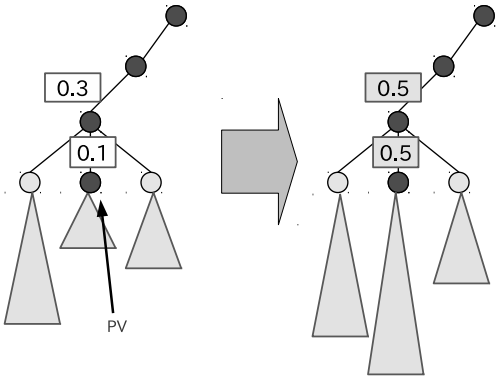

図 4　分散激指での PV 再探索

す。この再探索は全ての局面について行われる。図 3 では、左側の実現確率 0.1 の子局面を、右側のように実現確率 0.5 を与えて再度深く探索し直している。再探索は、実現確率打ち切りアルゴリズムにおいて強さを実現するために重要な要素である。

分散激指において、マスタ部分が持つ root 近傍の探索木については、浅い読みで作られた静的な木を利用しており、そのままでは実現確率の小さい (読みの浅い) 手順を最善手として回答してしまう。今回の実装でも、そのままではプレイヤの強さが大きく損われてしまうことが確認された。そこで、葉局面の評価値が得られたところで静的な木を部分的に更新し、再探索の効果を再現することを試みた。

葉局面の評価値を用いて min-max 木を作成すると、最善応答手順 (PV) が 1 つ求められる。図 4 の左側では、探索結果として PV が求まった時点が示されている。ここで、PV 中の経路についてのみ、実現確率を最善手である場合に相当する大きなものに変更し、再度葉局面からの探索を実行する。PV の手順の長さが全て最善手である場合相当に充分長くなるか、PV 手順に関わる実現確率が変化しなくなるまでこの手順を繰り返す。図 4 に具体例を示す。図の左側、最初の探索木では PV の経路は 0.3 や 0.1 のような小さな実現確率であったが、これを右側のように全て 0.5 に変更する。これにより、葉局面での実現確率が変更され、PV 周辺の局面はより深く探索されることになる。

現時点での実装では、PV の実現確率を変更した後、全ての葉局面について再探索を行っている。PV に関わらない葉局面は打ち切り閾値の変更がないので、これは無駄が多い。また、PV に関してのみ再探索を実施している点も、本来のアルゴリズムとは異なる。マスタ激指の持つ探索木中の全ての局面に対して、最善子局面への実現確率を大きくし、再探索を行うことは可能である。これらについては今後実装の改善を考えている。

なお、ワーカ部分の探索は激指の持つ探索アルゴリズムをそのまま利用しており、再探索も行われているため問題は生じない。

3.5 評　　価

分散化した激指を用いて、探索が高速化されるかを確認した。総タスク数は局面によって異なるが、おおむね 300 から 10000 程度になるように閾値を調整した。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。詳細に評価実験を行うとともに、今後実装を改善し、より強いプレイヤの実現を目指したいと考えている。

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を図ろうと考えている。

4. 関 連 研 究

4.1 GPS 将棋

GPS 将棋[7] は金子、田中らにより開発された将棋プレイヤであり、2009 年の世界コンピュータ将棋選手権で優勝した、トップレベル将棋プレイヤの 1 つである。田中らは、GPS 将棋において分散計算を適用し、300 ノード以上の大規模分散環境での探索を実現

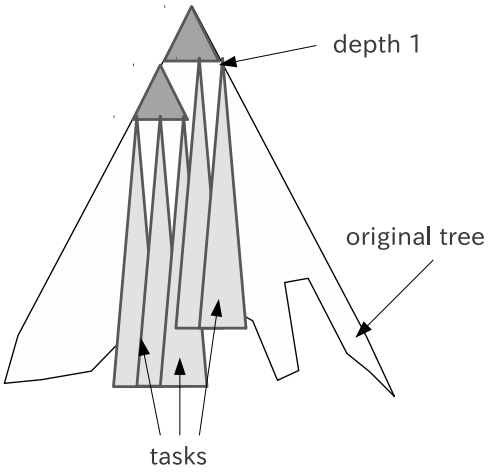

図 5 GPS 将棋のタスク分割

した[8]。GPS 将棋は、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、というアプローチをとった。

GPS 将棋の分散探索アルゴリズムは、以下の通り。

(1) 初期局面において短時間 (1 秒) 探索を行い、その局面の指し手候補に対し順序付けを行う。
(2) 上位の候補手いくつかについては探索木を延長し、残った候補手はその局面を葉として扱うこととする。(**図 5** の depth 1 の部分)
(3) 探索延長された局面についてこの手順を繰り返し、初期局面を根として有望な局面が延長された木を作成する。
(4) 葉の数が利用できる計算機数に達した時点で木の生成を停止する。全ての葉局面に計算機を割り当て、それぞれの局面から独立して探索を行い、葉局面の評価値を求める。(図 5 の tasks の部分)
(5) 各葉局面の評価値が変更されるたびに、探索木のミニマックス値を更新する。
(6) (一手の時間制限など) 探索終了条件が満たされた時点で、得られている探索木の評価値と最善手に従ってゲームをプレイする。

GPS はマスタワーカ型の制御構造をとっており、root 近くの探索木を静的に構築した後で、葉局面を独立したタスクとして分散計算するというアプローチをとっており、今回の激指分散化では同様の手法を用いた。

GPS は、計算に参加するノード数と同数の葉局面を生成し、1 つの計算ノードが 1 つの葉局面を探索終了時まで継続して計算しつづける、という手法を用いている。葉局面の難しさの違いや計算ノードの性能差によって、葉局面の探索の進捗は互いに異なることが予想され、元々の探索アルゴリズムで得られていた探索木とは大きく異なる探索を行ってしまうと考えられる。また、再探索についても考慮されていないため、浅い読みの葉局面からの探索が PV として採用される可能性がある。この問題に対し GPS では、浅い読みの葉局面については性能の高い計算ノードを割り当てることで対処しているが、どの程度改善されるかは議論を要する。

5. おわりに

本論文では、コンピュータ将棋プレイヤ「激指」に分散計算を適用し、近年広まりつつあるクラウドコンピューティング環境上で、探索性能を向上させることを狙った事例を紹介した。分散化に際しては、プログラムの変更をできるだけ少なくし、簡単に分散計算を実現できるような枠組みを採用することを試みた。分散計算処理系として GXP make を用いることで、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。

今後は、分散化のアルゴリズムを改良し、強いプレイヤを実現することを目指そうと考えている。葉局面においても探索の途中結果を用いて $\alpha\beta$ ウィンドウを設定し、無駄な探索を省くこと、transposition table の情報を一部共有するなどして局面の重複を排除すること、などの改善が考えられる。また、実現確率更新時の再探索に関して、分散計算アルゴリズムの改良も行っていきたい。

また、GXP make について、動的に生成されるタスクを扱う機能が不足していることも明らかになった。これについても、処理系の機能拡張などで対応していきたいと考えている。

参 考 文 献

1) Amazon: Amazon Elastic Compute Cloud. http://aws.amazon.com/jp/ec2/.
2) Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *OSDI'04: Sixth Symposium on Operating System Design and Implementation* (2004).
3) Google: Google App Engine. http://code.google.com/intl/ja/appengine/.
4) Isard, M., Budiu, M., Yu, Y., Birrell, A. and

Fetterly, D.: Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks, *European Conference on Computer Systems (EuroSys)* (2007).
5) Taura, K.: GXP : An Interactive Shell for the Grid Environment, *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp. 59–67 (2004).
6) Tsuruoka, Y., Yokoyama, D. and Chikayama, T.: Game-tree Search Algorithm based on Realization Probability, *ICGA Journal*, Vol. 25, No.3, pp.145–152 (2002).
7) Game Programming Seminar: GPS 将棋. http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/.
8) 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).

質疑応答

Q (大座畑) 将棋の対局は人間とやるのか?コンピュータ同士を想定しているのか?

A どちらもある。1 年に 1 度行われているコンピュータ将棋選手権ではコンピュータ同士が対局する。製品版では人間との対局を想定して作成している。また、2010 年 10 月には情報処理学会のイベントで、女流プロのトップレベルである清水女流とコンピュータが対局する。

Q (大座畑) チェスではカスパロフの癖を使っていたが、その場合カスパロフ以外のグランドマスターには負けるのではないか? 激指はそういったヒューリスティックスはどれくらい入っているのか?

A 激指の場合、プロの棋譜から評価関数と読みのコントロール部分 (どこを深く読むか、という制御) について、人間の指した棋譜を正しいものとして、学習している。対局相手を想定したヒューリスティックはまだ入れていないが、相手のモデルを反映した探索は今後の研究課題の一つだろう。

Q (大座畑) 現状は、人間と比べたらどれくらいなのか? 羽生さんのような将棋のトッププレイヤとは戦っていないのか?

A プロとは将棋連盟に禁止されている。トップアマとはいい勝負。

Q (大座畑) コンピュータ将棋選手権しか対局していないのか? プログラムの中で強いだけなのか?

A 人間のトップレベルアマチュアとは対局している。現時点ではアマチュアトップと同じくらいの対局成績。ただし、数年前の竜王との対局では相当強さに差がある感じで負けた。

Q (副田) こういった探索とクラウドは相性が悪い気がする。モンテカルロ碁のような方法は適用できないのか?

A 今のところ、モンテカルロ将棋はうまくいっていない。

Q (副田) 収穫逓減はどの程度か?

A 速度向上の限界点は比較的早く来ると考えている。とはいえ、1 万台程度投入すれば、non-cloud 版に勝てるかもしれない。

Q (岩崎) GXP を使ったアプリケーションにはどのようなものがあるか?

A 大規模なものとしては、医療分野の大量文書を対象とした自然言語処理がある。

Q (岩崎) それぞれの仕事の独立性は高いのか?

A ファイル毎に独立性が高いようなタスク。1 つのファイルに対する処理中には依存関係はある。

Q (並木) 今後はがりがりハードコーディングするのか? それとも，もうちょっと GXP で頑張るのか?

A GXP に関してはこの辺で、今後はハードコーディングする。

Q (並木) GXP アプローチはこれ以上やらないのか?

A 考察したように、GXP に足りない部分などを提供していくことは可能そうだと思う。

Q (並木) 下のレイヤ構造も踏まえて、ロードバランシングとかはしないのか?

A タスクに対するアノテーションなどを入れて、スケジューリングに役立てるアプローチはあると考えている。