Vol.2011-HPC-130 No.59
2011/7/29

# 分散制約充足問題のジョブ並列による求解

安部達也[†1] 平石拓[†2] 三宅洋平[†3]
岩下武史[†2] 中島浩[†2]

分散制約充足問題を分散並列計算環境で解くにあたり，ジョブを処理の単位とする分散並列処理（ジョブ並列）に特化したジョブ並列スクリプト言語 Xcrypt で処理を記述することにより，実際の分散並列計算環境であるところの，いわゆるスーパーコンピュータを利用する方法を紹介する．さらに，Xcrypt の遠隔ジョブ投入機構を利用することにより，制約が遠隔の計算機に分散された状態からの制約充足問題，つまり，真の意味での分散制約充足問題を簡便に取り扱うことができることを示す．

# Job-level Parallel Executions for Satisfying Distributed Constraints

Tatsuya Abe,[†1] Tasuku Hiraishi,[†2] Yohei Miyake,[†3]
Takeshi Iwashita[†2] and Hiroshi Nakashima[†2]

We introduce a method of parallel executions based on the job unit (job-level parallel executions) for solving distributed constraint satisfaction problems (DCSPs) in parallel and distributed computation environments, the so-called today's many supercomputers. Throughout introducing the method we use the job-level parallel script language Xcrypt, specific to job-level parallel executions. We also show that Xcrypt provides us with a feature of submitting remotely jobs for solving realistic DCSPs (under the circumstances that constraints are truely distributed in separate computers).

†1 理化学研究所計算科学研究機構
Advanced Institute for Computational Science, RIKEN
†2 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University
†3 神戸大学大学院システム情報学研究科
Graduate School of System Informatics, Kobe University

## 1. はじめに

分散制約充足問題とは，制約と呼ばれる（一般には複数の）条件を同時に満たすものを発見する問題である，いわゆる制約充足問題の一種であり，特に，エージェントや制約が分散されているものをいう[10]．この問題を考察する理由には，到来した分散並列計算環境を利用するにあたってエージェントが分散されていることは当然の仮定であること，また，プライバシの観点から，制約は分散されていると仮定する（制約が全エージェントで共有されていると仮定すべきでない）ことが自然であることが挙げられる．

分散制約充足問題を解くにあたり，そのアルゴリズムの研究は非常に活発であり[2),4),6)]，そのようなアルゴリズムを評価するためのベンチマークのデファクトスタンダード（Sensor-DCSP）も与えられており，今後も分散制約充足問題のためのアルゴリズムはますます活発に研究される傾向にある[1)]．

その一方，研究・開発されたアルゴリズムを実装するにあたっては，Java フレームワーク FRODO[⋆1] を利用することが通例である[3)]．FRODO は分散制約充足問題のソルバを，Communication Layer，Solution Spaces Layer，Algorithm Layer と呼ばれる階層ごとに実装できる機構を提供しており，分散制約充足問題を解くためのアルゴリズムの試行実装に適している．また，前述の SensorDCSP を内蔵していることや新しく発見されたアルゴリズムが次々と FRODO 上に実装されていくことからもアルゴリズムを実装・評価するためのフレームワークとして FRODO を利用するのがよいと考えられる．

本稿では，実際に分散制約充足問題を解くために分散並列計算環境を利用するにあたって，分散されたデータ（制約）の扱いと実行コードの分散の防止に焦点をあてる．実際の分散並列計算環境（通常はスーパーコンピュータ）を利用するにあたっては，手元の計算機を利用するのとは異なり，例えば，バッチジョブスケジューラを通して利用しなければならないといった，そのスーパーコンピュータシステムの流儀に従う必要がある．そこで本稿では，分散制約充足問題を解くためにそのようなシステムを利用するにあたって，逐次実行可能なプログラム（例えば FRODO Version 2.7.3 の 4.3 節 Simple Mode に挙げられているプログラム）をスーパーコンピュータシステムの流儀に従って分散並列実行させることで，分散制約充足問題を真の意味で分散・並列に解く．スーパーコンピュータシステムの流儀に従うにあたってはそれに特化した，いわゆるドメインスペシフィック言語であるところの，ジョ

⋆1 http://liawww.epfl.ch/frodo/

ブ並列スクリプト言語 Xcrypt を利用する[12]．

本研究は，FRODO Version 2.7.3 の 4.4 節 Advanced Mode に述べられている動機を実際のスーパーコンピュータの利用へ向けたものと考えられる[3]．

本稿の構成は以下である．2 節で今回実装例として扱う Petcu と Faltings の DTREE アルゴリズムの紹介をする[5]．3 節で DTREE の実装に使用するジョブ並列スクリプト言語 Xcrypt の紹介をする．4 節で Xcrypt による DTREE アルゴリズムの実装を詳説する．5 節では遠隔ジョブ投入を行う方法を紹介する．6 節では関連研究と本研究がどのような発展をする可能性があるかについて述べる．

## 2. DTREE アルゴリズム

分散制約充足問題とは，以下の三つ組 $\langle X, D, R\rangle$ をいう．

(1) エージェントの集合 $X = \{x_i \mid 0 \le i \le m\}$ は変数の集合であり，

(2) 領域の集合 $D = \{d_i \mid 0 \le i \le m\}$ は変数がとり得る値の集合の集合であり，

(3) 制約の集合 $R = \{r_i \mid 0 \le i \le n\}$ は領域の組から $\mathbb{R} \cup \{\infty, -\infty\}$ への関数の集合である．

変数はその変数への値の割り当てを計算するエージェントと単位を同じくしている．$x_i$ へ割り当てられる値は $d_i$ から選ばれる．$r: d \times d' \to \mathbb{R} \cup \{\infty, -\infty\}$ は，ある変数 $x$ が値 $v \in d$ をまたある変数 $x'$ が値 $v' \in d'$ をとった時のユーティリティを意味している．この定義の下で，分散制約充足問題は，すべてのエージェントのユーティリティの総和を最大にする変数への値の割り当て，つまり，$X$ から $\prod\{d_i \mid 0 \le i \le m\}$ への関数を見つける問題であると定式化される．

割り当てを見つけるエージェントは一般にはグラフ構造を成していると考えられる．しかし，本稿では，簡単のため，割り当てを見つけるエージェントは木構造を成していると仮定する．また，近傍（親と子たち）のエージェントのみと制約を共有していると仮定する（そのような木構造であると仮定する）．

DTREE は Petcu と Faltings により考案された，木構造を成しているエージェントたちへの値の割り当てを見つけるアルゴリズムである[5]．DTREE では，エージェントは自身への値の割り当てを決定するため，近傍のエージェントのみと通信を行う．このように，放送を行わないことでプライバシの問題を回避している．DTREE の手順の概略は以下である．

(1) 葉のエージェントは，親のエージェント $x_i$ に自身が持つ制約から見積もられるユーティリティを（$d_i$ が離散であればベクタで，連続であれば $d_i$ からの関数で）送る．この送られるメッセージを UTIL メッセージと呼ぶ．

(2) 一般のエージェントは子のエージェントの UTIL メッセージを待つ．それらをすべて受けとった時，自身を根とする部分木で見積られる UTIL メッセージを親のエージェントに送る．

(3) 根のエージェントは子のエージェントの UTIL メッセージを待つ．それらをすべて受けとった時，自身を根とする木でユーティリティを最大にする，自身への値の割り当てを決定する．この自身への値の割り当てを子のエージェントに送る．このメッセージを VALUE メッセージと呼ぶ．

(4) 一般のエージェントは親のエージェントから VALUE メッセージを受け取る．その下で，ユーティリティを最大にする，自身への値の割り当てを決定し，VALUE メッセージを子のエージェントに送る．

(5) 葉のエージェントは親のエージェントから VALUE メッセージを受け取る．その下で，ユーティリティを最大にする，自身への値の割り当てを決定する．

この手順でユーティリティの総和を最大にする変数への値の割り当てが得られることの詳細は Petcu と Faltings の原典に譲る[5]．本稿では，この手順を実際に行う際，特に，実際の分散並列計算環境で行う際に問題となることを議論したい．まず，DTREE アルゴリズムは以下の 2 つの特徴を持つ．

- エージェントは，自身が木のどのノードであるか（根であるか葉であるかそのいずれでもないか）によって動作を異にする．
- エージェント間で通信されるものは UTIL メッセージと VALUE メッセージのみであり，プライバシの観点より制約が通信されることはない．

この 2 つの特徴より，一見すると各エージェントが各データ（制約）を保有するのと同様に各エージェントが各実行コードをも保有するのが自然であるように思える．一方，コードの正しさや保守のしやすさという点から，一つの手順を実行するコードをエージェントに分散保有させることは好ましくないとも考えられる．本稿では後者の立場から，データ（制約）はエージェントが保有する一方，実行コードはある計算機（エージェントでなくてもよい）が一元的に管理し，エージェントに実行させたい処理があればその都度その処理を行うコードをそのエージェントに渡すという方法をとる．次節以降，この方法をとるにあたり，ジョブを処理の単位とする分散並列計算に特化したジョブ並列スクリプト言語 Xcrypt を利用することで，スーパーコンピュータを容易に利用できるようになることと，それに加えて，ユーザのプログラム記述負担を軽減させることができることを 4 節で紹介する．

Vol.2011-HPC-130 No.59
2011/7/29

## 3. ジョブ並列スクリプト言語 Xcrypt

本節では，ジョブを処理の単位とする分散並列計算に特化したジョブ並列スクリプト言語 Xcrypt を紹介する[12]．

一般的な分散並列計算環境において，プログラムの起動は単純なコマンド実行では行えず，ジョブスクリプトを記述した上でバッチジョブスケジューラに対してジョブを投入するということを必要とすることが多い．また，多数のジョブを非同期に実行し，それらのジョブの完了を待ち合わせたいという要望も多い．しかし，これらの処理を Perl などのスクリプト言語で直接記述することは非常に手間がかかる．それに加えて，バッチジョブスケジューラとユーザとのインターフェースがシステムごとに異なることはそのようなスクリプトの可搬性を下げてしまうという問題を持っている．これらの問題を解決するためには，処理を記述するのに使用する言語がスクリプト言語であることに加え，ジョブ並列処理に関する共通の処理に対してライブラリ等による支援を行える言語であるのがよい．Xcrypt は，既存のスクリプト言語である Perl をベースとし，これにジョブ並列処理の簡便な記述のために必要となる様々な支援機能を追加して開発されているジョブ並列スクリプト言語である．Xcrypt を用いることにより，ユーザは単純なパラメータスイープから複雑な最適パラメータ探索アルゴズムまでの様々な処理を，通常のシェルスクリプト等によるコマンド実行に近い感覚で記述することができる．Xcrypt ではまた，複雑な探索アルゴリズムやジョブの同時投入数の制限などのジョブ並列処理において共通に用いられる様々な機能をライブラリとして提供することができる機構を持つ．それらのライブラリを取り込むだけで Perl に不慣れなユーザでも複雑な機能を利用できる．

Xcrypt スクリプトの一例を**図 1** に挙げる．このスクリプトを通して Xcrypt の概略を説明する．

1 行目の `use` 文は Perl においてモジュールを読み込むための文である．`core` モジュールは Xcrypt のコアモジュールであり，Xcrypt スクリプトの先頭行は `use base qw(`$module_0 \ldots module_{l-1}$ `core);` であると決められている．ここで $module_0 \ldots module_{l-1}$ は任意の Xcrypt モジュールである．`sandbox` モジュールについては 4 節で詳しく説明する．

2 行目は Perl の `print` 関数である．この `print` 関数の出力であるメッセージは Xcrypt を実行した計算機の標準出力，つまり，眼前のディスプレイに表示される．

3 行目は Xcrypt の `spawn` 文であり，この spawn ブロック内で記述された処理を行うオブジェクト（ジョブオブジェクトと呼ぶ）を生成する．ジョブオブジェクトで行われる処理は

```
01: use base qw(sandbox core);
02: print "Printed on the log-in node.";
03: spawn {
04:  print "Stored in the standard output on the computational node.";
05:  qx('./a.out');
06: };
07: sync;
```

**図 1** Xcrypt スクリプトの一例
Fig. 1 A sample script in Xcrypt

（Xcrypt を実行した計算機とは限らなく）ジョブを処理する計算機で実行される．ジョブを処理する計算機の特定はシステム設定やユーザ設定，Xcrypt をコマンドライン実行した際のオプションの与え方，あるいは Xcrypt スクリプト内の記述でなされている．このように Xcrypt はユーザにジョブを処理する際に必要となる設定を多様な方法で可能とする機構を提供している．ユーザは，自身に最適な設定方法を選択すること，本例でいえば，Xcrypt スクリプトにジョブ情報が現れないようにすることで，この Xcrypt スクリプトに可搬性を与えることができる．

図 1 には記述されていないが，`spawn` 文にはそのジョブの前処理と後処理とを記述するための`_before_`ブロックと`_after_`ブロックとをそれぞれ続けて書くことができる．これらのブロック内に書かれた処理はジョブ投入の直前とジョブ終了の直後にジョブを投入する計算機でそれぞれ並行に実行される．

4 行目は Perl の `print` 関数である．この `print` 関数の出力であるメッセージはジョブを処理する計算機の標準出力に出力される．多くの場合はバッチジョブスケジューラにデフォルトで設定されているファイルに書き出される．そのために（当然であるが）眼前のディスプレイにこのメッセージは表示されない．

5 行目は Perl の `qx` 関数である．この `qx` 関数も 4 行目の `print` 関数と同様にジョブを処理する計算機で実行されるため，`"./a.out"`というプロセスはジョブを処理する計算機で立ち上げられる．

7 行目は Xcrypt の `sync` 文である．これは `spawn` 文で生成されたジョブオブジェクトの処理を待ち合わせる．本例では，`spawn` 文で生成されたジョブオブジェクトは 1 つなので，

```
01: use base qw(sandbox core);
02: create_tree();
03: foreach $agent (0..6) {
04:  spawn {
05:   qx('../compute_utils');
06:  }_before_{
07:   mkdir 0755, $agent;
08:   rename config_$agent, $agent;
09:   ($p, @cs) = dtree::get_parent_children();
10:   dtree::wait_for_util_message(@cs);
11:  }_after_{
12:   dtree::send_util_message($agent, $p);
13:   dtree::wait_for_val_message($p);
14:   dtree::create_val_message($agent, $p);
15:   dtree::send_val_message($agent, @cs);
16:  } ('id' => $agent, 'header' => ['use dtree;']);
17: }
18: sync;
```

図 2　DTREE の Xcrypt スクリプトの一例
Fig. 2　A sample script of DTREE in Xcrypt

そのジョブオブジェクトの処理を待つ．

このように Xcrypt は，手元の計算機での処理とジョブを処理する計算機での処理とが混在する，それでいて，それらの処理の切り分けがユーザに容易である記述（spawn ブロック内であるかそうでないか）を許す．このことがコードの正しさや保守のしやすさを提供することは前述した通りである．4 節では 2 節で紹介した DTREE アルゴリズムを実際に Xcrypt スクリプトで記述する．

## 4. DTREE の Xcrypt によるジョブ並列実行

2 節で紹介した DTREE アルゴリズムを 3 節で紹介した Xcrypt で記述したスクリプトの一例を図 2 に載せる．

1 行目では use base qw(sandbox core); で sandbox モジュールを読み込んでいる．通常の Xcrypt 実行では作業ディレクトリはカレントディレクトリであり，Xcrypt スクリプト処理中に作成されるファイルは作業ディレクトリ，つまり，カレントディレクトリからの相対パスで指定された場所に置かれるが，sandbox モジュールが読み込まれている場合，ジョブオブジェクトの生成時にジョブ固有の識別子（ID）を作成し，その ID を名前とするディレクトリをそのディレクトリが存在しなければカレントディレクトリからの相対パスで作成し，そのディレクトリを作業ディレクトリとする．これにより，並列に実行されるジョブが作成するファイル間に名前の衝突が起こらない．

2 行目では関数 create_tree() を呼び出している．この関数は，制約の依存関係を把握している計算機にエージェントたちをノードとする制約グラフ（本稿ではこれが木になることを仮定している）を生成させる．また，その木における親子関係，変数，その変数がとり得る値（つまり領域），制約を格納したプロパティファイル形式のファイルを config_$agent という名前で生成する．制約の依存関係を把握している計算機を仮定しない場合，つまり，エージェントからなる木構造が静的に決定しており，適切な config_$agent ファイルを各エージェントが保有していると仮定できる場合には，この関数を呼び出す必要はない．

3 行目の Perl の foreach 関数で自然数 $0, 1, 2, 3, 4, 5, 6$ を名前とする 7 体のエージェントを定義している．

4 行目の spawn 文でジョブオブジェクトを生成している．この spawn ブロックは foreach ブロック内にあるので計 7 つのジョブオブジェクトが生成される．その各ジョブオブジェクトに与えられている引数は 16 行目の ('id' => $agent, 'header' => ['use dtree;']) という Perl ハッシュ（連想配列）であり，スカラ変数$agent の値を ID とするジョブオブジェクトであり，その処理をする Perl スクリプトのヘッダで dtree モジュールを読み込むということを表している．この spawn ブロック内で記述されている処理はそのジョブが投入される計算機で行われる．また，6 行目の_before_ブロック内に記述されている処理はそのジョブを投入する計算機でジョブ投入の直前にジョブオブジェクトごとに並行に実行される．同様に，11 行目の_after_ブロック内に記述されている処理はそのジョブを投入する計算機でジョブ終了の直後にジョブオブジェクトごとに並行に実行される．

5 行目では外部プロセス compute_utils を呼び出している．このプロセスはエージェントが制約からユーティリティを最大にする割り当てを決定するための処理を行うものであり，外部プロセスであることからわかるように，必ずしも Xcrypt や Perl で記述されたスク

リプトでなくてもよく，一般のプログラミング言語で記述されたプログラムでよい．このようにしているのは，分散制約充足において重い処理を行うであろう制約充足部分にプログラミング言語の制限をかけるべきではないという考えからであり，処理自体は重くない通信部分においては Xcrypt や Perl での記述を仮定しても，これらの言語は手軽に記述できることを優先して設計されているから問題ないという考えによる．エージェントはユーティリティを計算する一方，最終的に割り当てを決定するために必要になる情報をまとめたファイルをここで作成・保存しておく．

7 行目の Perl の mkdir 関数で$agent という名前のディレクトリを生成している．

8 行目の Perl の rename 関数で create_tree が生成した config_$agent を$agent ディレクトリに移動している．

9 行目では dtree モジュールで定義されている関数 get_parent_children() を呼び出している．この関数は，自エージェントが保有しているエージェントの親子関係を記載しているファイルを読み込むことで，唯一の親エージェント（エージェントは木構造を成していると仮定している）または文字列 null（そのエージェントが根であることを意味している）と 0 体以上の子エージェントを返す．

10 行目では dtree モジュールで定義されている関数 wait_for_util_message を引数@cs で呼び出している．@cs は 0 体以上の子エージェントであり，この関数はすべての子エージェントから UTIL メッセージが送られてくるまでブロックする．自エージェントが葉エージェントであれば子エージェントを持たないということなので，待つことなくただちに次の行の処理に移る．

12 行目では dtree モジュールで定義されている関数 send_util_message を引数$agent と@cs で呼び出している．これは 5 行目で計算し作成した UTIL メッセージを親エージェントに送ることを表している．自エージェントが根エージェントである場合は親エージェントを持たないということなので，何もせずただちに次の行の処理に移る．

13 行目では dtree モジュールで定義されている関数 wait_for_val_message を引数$p で呼び出している． $p は唯一の親エージェントまたは文字列 null であり，この関数は親エージェントから VALUE メッセージが送られてくるまでブロックする．自エージェントが根エージェントであれば親エージェントを持たないということなので，待つことなくただちに次の行の処理に移る．

14 行目では dtree モジュールで定義されている関数 create_val_message を引数$agent と$p で呼び出している．$p は唯一の親エージェントまたは文字列 null であり，この関数は（親エージェントがいるなら）親エージェントから送られてきた VALUE メッセージから子に送る VALUE メッセージを作成する．

15 行目では dtree モジュールで定義されている関数 send_val_message を引数$agent と@cs で呼び出している．@cs は 0 体以上の子エージェントであり，この関数はすべての子エージェントへ VALUE メッセージを送る．自エージェントが葉エージェントであれば子エージェントを持たないということなので何もしない．

18 行目の sync 文で投入されたジョブの待ち合わせを行っている．

UTIL メッセージと VALUE メッセージはファイルの複製・転送によってやりとりされる．これはスーパーコンピュータシステム利用を前提としていることによる．というのも，スーパーコンピュータシステムでは TCP/IP 等による通信が可能であることを一般的には仮定できない一方，NFS 等のファイル共有システム，そうでなくても，ステイジングによる間接的なファイル共有の仕組みが存在することは仮定できるからである．

## 5. 遠隔ジョブ並列実行

前節までは，分散制約充足問題を 1 つの計算機システムで解く方法について解説してきた．本節では，異なるサイトに存在する複数の計算機を使って 1 つの分散制約充足問題を解く方法を紹介する．これにより真の意味で分散する制約を充足する解を求めることになる．

異なるサイトに存在する複数の計算機を利用するには，その計算機を利用するための認証を通過することとその計算機で実行することに関する記述が，前節で準備したものに追加で必要である．現版の Xcrypt は認証にあたって Fandiño による Perl モジュール Net::OpenSSH[⋆1] を利用しているため，OpenSSH[⋆2] を採用している多くのスーパーコンピュータでの利用に際して，認証のためだけに新たな機構の導入を強制しない．Xcrypt は認証に関する設定も他の設定と同様にユーザに多様な方法を提供する．一般に，認証方法やその設定は頻繁に変更されるものではないため，Xcrypt スクリプト中で設定を変更することを動的な設定と呼ぶことにすると，今回は，認証方法やその設定を静的に設定しているものと仮定する．つまり，OpenSSH 設定に関する情報は後述する Xcrypt スクリプトに現れない．

遠隔の計算機に頻繁にコマンドを発行するにあたり，その ssh セッションを維持する機構を Xcrypt は自前で有しておらず，Perl モジュール Net::OpenSSH に依存している．そのた

⋆1 http://search.cpan.org/dist/Net-OpenSSH/
⋆2 http://www.openssh.org/

```
01: use base qw(sandbox core);
02: $env0 = get_local_env();
03: $env1 = add_host({'host' => 'foo1@bar1', 'sched' => 'sge'});
04: $env2 = add_host({'host' => 'foo2@bar2', 'sched' => 'torque'});
05: $env3 = add_host({'host' => 'foo3@bar3', 'sched' => 'condor'});
06: @queue = ('occupied', 'common', 'occupied_xxx', 'preferential');
07: @env = ($env0, $env1, $env2, $env3);
08: @job = ();
09: foreach $agent (0..3) {
10:  ($self) = spawn {
11:   qx('../compute_utils');
12:  }_before_{
13:   ($p, @cs) = dtree::get_parent_children();
14:   dtree::wait_for_util_message(@cs);
15:  }_after_{
16:   dtree::send_util_message($agent, $p);
17:   dtree::wait_for_val_message($p);
18:   dtree::create_val_message($agent, $p);
19:   dtree::send_val_message($agent, @cs);
20:  } ('id' => $agent, 'header' => ['use dtree;'],
21:      'env' => $env[$agent], 'JS_queue' => $queue[$agent]);
22:  push(@job, $self);
23: }
24: async {
25:  while(1) {
26:   dtree::transfer_message(@job);
27:   Coro::AnyEvent::sleep 10;
28:  }
29: };
30: sync;
```

**図 3** 遠隔ジョブ投入を行う Xcrypt スクリプトの一例
Fig. 3 A sample script of remotely job submissions in Xcrypt

めに Xcrypt は Net::OpenSSH が有していないが本家の OpenSSH クライアントが有している ssh の機能，多段 ssh（リモートログインした計算機からさらにリモートログインすること）を提供していない．しかし，今回扱う DTREE や 6 節で紹介するその亜種は多段 ssh を必要としない[⋆1]．

図 2 の Xcrypt スクリプトを遠隔ジョブ投入用に書き直した Xcrypt スクリプトが**図 3** である．図 2 の Xcrypt スクリプトと重複する部分の説明を避け，差異を説明する．

2 行目では Xcrypt 関数 `get_local_env()` を呼び出している．この関数は Xcrypt を実行した計算機のバッチジョブスケジューラの Xcrypt における情報をまとめたものを Perl ハッシュのリファレンスで返す．これまではこのハッシュリファレンスを明示的に扱わなくても，Xcrypt がデフォルトでこのハッシュリファレンスをジョブオブジェクトに埋め込んでいたため問題なかった．しかし，遠隔ジョブ投入を行う際には，どのジョブをどのサイトに存在する計算機で処理するかを明示的に扱う必要があるため，このように `get_local_env()` を呼び出してデフォルトである Xcrypt を実行した計算機のバッチジョブスケジューラの情報を取得している．

3–5 行目では Xcrypt 関数 `add_host` を呼び出している．この関数は引数で与えられた計算機のバッチジョブスケジューラの情報を元に，それを Xcrypt で扱える Perl ハッシュのリファレンスにして返す．引数として与えられる Perl ハッシュのリファレンスは最低でもホスト名@ドメイン名とその計算機のバッチジョブスケジューラの情報を含んでいなければならない．サンプルスクリプト中の `sge`，`torque`，`condor` はそれぞれ代表的なバッチスケジューラである Sun Grid Engine[⋆2]，Torque[⋆3]，Condor[⋆4] を意味している．

6 行目では各計算機でユーザが使用可能であるジョブキューを定義している．このサンプルスクリプトでは，ユーザの手元にある計算機のジョブキュー（手元の計算機であり占有されているであろうから）`occupied`，`bar1` のジョブキューで共有であるものと推察できる `common`，`bar2` のジョブキューで占有利用可能で識別子が `xxx` であるものと推察できる `occupied_xxx`，`bar3` のジョブキューで優先利用可能であるものと推察できる `preferential` が定義されている．

7 行目ではユーザが使用可能である計算環境情報を配列に詰めている．12 行目の `spawn`

⋆1 メッセージパッシングが成す木（グラフ）構造と ssh の木（グラフ）構造は別であることに注意する．
⋆2 http://gridengine.sunsource.net/
⋆3 http://www.clusterresources.com/pages/products/torque-resource-manager.php
⋆4 http://www.cs.wisc.edu/condor/

でジョブを投入する際，この配列から計算環境情報を取得する．

10 行目では spawn 文の返り値を取得をしている．返り値はジョブオブジェクトの配列である．

20–21 行目では spawn 文の引数を与えている．今回，ジョブを処理する計算機が 1 つではないため，その情報を Xcrypt のジョブオブジェクトのメンバ env と JS_queue とすることでジョブに与えている．

24 行目では Xcrypt が利用している Lehmann による Perl モジュール Coro[⋆1]の関数 async である．この async ブロック内の処理はメインスレッドと並行に実行される．

26 行目では dtree モジュールで定義されている関数 dtree::transfer_message() を引数@job で呼び出している．この@job は 10 行目で取得したジョブオブジェクトの配列である．通常，複数のサイトに存在する計算機間でデータを送りあう際には，スーパーコンピュータシステムの利用制限のため，手元の計算機を介しての 3 点移動を行わなければならない．そこで，この関数は UTIL メッセージと VALUE メッセージを監視し，それの受け渡しをビジーウェイトで実現している．具体的には，制約木を生成した際に親子関係を記録していたファイルからエージェントの位置情報を取得し，それとジョブオブジェクトの配列 @job を利用してこれら 2 種類のメッセージを各エージェントに配送している．

## 6. おわりに

本稿では，分散制約充足問題を実際のスーパーコンピュータで解くにあたり，ジョブ並列とそれに特化したジョブ並列スクリプト言語 Xcrypt を使う方法を紹介した．ジョブ並列を積極的に利用するという考え方は新しい考え方であるため，本研究に類似する文献は見当たらない．しかし，スーパーコンピュータを利用するにあたってしばしば必要とされるバッチジョブスケジューラを介する利用を煩わしいとする考え方は存在しており，その煩わしさを軽減するためのツールは存在する[9),11),13),14)]．本研究で行ったことが，これらのツールでどのように実現できるかは今後の研究課題である．

本稿で利用した Xcrypt においても，本研究で行ったことの実現にスクリプトの記述という点で課題が残った．Xcrypt では，ジョブとして行いたい処理を単に spawn ブロック内に記述するだけでよい．それゆえに当初は今回行いたい処理を **図 4** と書けると期待していた．このように書けるとすると，図 2 と同様にエージェントごとに処理がまとまっているだけで

⋆1 http://search.cpan.org/dist/Coro/

```
spawn {
 ($p, @cs) = dtree::get_parent_children();
 dtree::wait_for_util_message(@cs);
 qx('../compute_utils');
 dtree::send_util_message($agent, $p);
 dtree::wait_for_val_message($p);
 dtree::create_val_message($agent, $p);
 dtree::send_val_message($agent, @cs);
} ('id' => $agent, 'header' => ['use dtree;']);
```

**図 4** デッドロックの危険を持つオブジェクト集合
Fig. 4 A set of objects reaching a deadlock

なく，図 2 と異なり本処理（spawn ブロック内に記述されているもの）が前処理（_before_ ブロック内に記述されているもの）に先行しない．というよりも，そもそも前処理・後処理の区別なく spawn ブロック内の記述だけで済むという利点がある．しかし，このスクリプトはジョブ内で待ち合わせを行っているために，あるジョブの終了を待っているジョブが資源を解放しないことにより，実はその待たれているジョブがいつまでたっても終了しない状態，いわゆるデッドロック状態に陥る危険を孕んでいる．この危険を回避する候補の一つに，ジョブが一時的に資源を解放することができる仕組みを導入するということが考えられる．そのような仕組みを計算機システム自体が持っていればそれでよいし，そうでなくても，例えば，継続を保存した後，一度ジョブを終了して，ときどき待ち合わせ状態を確認して，抜けられるようであればその継続により実行を再開するジョブを実行しなおしてみる，といったことを Xcrypt に行わせるというのでもよい．もし，そのような仕組みがあれば，この例でいうところの関数 wait_for_util_message の際にジョブが資源を他ジョブ（ジョブキューで待たされているものも含む）に譲ることができ，その結果，デッドロックの危険は回避される．この候補は提案として挙げておく．

Xcrypt はジョブがどのような状態（ジョブキューで待たされている・実行中である・実行を終了している等）であるか（ジョブ監視機構）について，スクリプトへの記述とコマンド実行というインタラクティブな操作との 2 つの方法をユーザに提供している．このジョブ監視機構はそのジョブ実行が遠隔ジョブ実行であるかどうかをユーザが意識する必要が無いよ

うにつくられている．本稿で扱った DTREE アルゴリズムでも，ユーティリティを計算する計算機（つまり，エージェント）がダウンしてしまった・計算機どうしをつないでいるネットワークが物理的にまたはソフトウェア的に切れてしまった等の障害が生じた場合でも，同 Xcrypt スクリプトを再実行すれば，障害が起きなかった場合と同じ結果が得られる．ただし，これには各エージェントが前回と今回とで同じ結果を返しているということが前提となっている．また，この前提が満たされているかを検知する処理も未実装である．この意味での耐故障性の保証は今後の研究課題である．

さらに，本稿で述べたジョブ並列実行手法は DTREE 以外のアルゴリズムに対しても考えることができる．DTREE では，エージェントたちが木構造を成していることを仮定しているが，一般にエージェントたちがグラフ構造を成していることのみ仮定している DPOP も同様に Xcrypt スクリプトを書くことでジョブ並列実行可能であると考えられる[6]．また，DPOP の亜種に対しても可能であると考えられる[7),8)]．

**謝辞**　本研究を行う動機といくつかの有益な情報を与えてくれた Xavier Olive 博士に感謝の意を表する．本研究の一部は，文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」の支援による．

## 参 考 文 献

1) Fernàndez, C., Béjar, R., Krishnamachari, B. and Gomes, C.: Communication and Computation in Distributed CSP Algorithms, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol.2470, Springer-Verlag, pp.664–679 (2002).
2) Hirayama, K. and Yokoo, M.: Distributed Partial Constraint Satisfaction Problem, *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol.1330, Springer-Verlag, pp.222–236 (1997).
3) Léauté, T., Ottens, B. and Szymanek, R.: *FRODO: a FRamework for Open/Distributed Optimization*, 2.7.3 edition (2011).
4) Modi, P.J., Shen, W.-M., Tambe, M. and Yokoo, M.: Adopt: asynchronous distributed constraint optimization with quality guarantees, *Artificial Intelligence*, Vol.161, No.1-2, pp.149–180 (2005).
5) Petcu, A. and Faltings, B.: A distibuted, complete method for multi-agent constraint optimization, *Proceedings of the 5th International Workshop on Distributed Constraint Reasoning*, pp.21–36 (2004).
6) Petcu, A. and Faltings, B.: DPOP: A Scalable Method for Multiagent Constraint Optimization, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pp. 266–271 (2005).
7) Petcu, A. and Faltings, B.: S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization, *Proceedings of the 12th National Conference on Artificial Intelligence*, pp.449–454 (2005).
8) Petcu, A. and Faltings, B.: O-DPOP: An algorithm for open/distributed constraint optimization, *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pp.703–708 (2006).
9) Taura, K.: GXP: An Interactive Shell for the Grid Environment, *Proceedings of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.59–67 (2004).
10) Yokoo, M., Durfee, E.H., Ishida, T. and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proceedings of the 12th International Conference on Distributed Computing Systems*, pp.614–621 (1992).
11) 富士通株式会社：Parametric Job Organizer 使用手引書，Fujitsu Parallelnavi Language Package 3 付属マニュアル edition (2008).
12) 平石　拓，安部達也，三宅洋平，岩下武史，中島　浩：柔軟かつ直観的な記述が可能なジョブ並列スクリプト言語 Xcrypt，第 8 回先進的計算基盤システムシンポジウム，pp. 183–191 (2010).
13) 大塚保紀，深野佑公，西里一史，大野和彦，中島　浩：タスク並列スクリプト言語 MegaScript の構想，第 1 回先進的計算基盤システムシンポジウム，pp.73–76 (2003).
14) 湯山紘史，津邑公暁，中島　浩：タスク並列言語 MegaScript における高精度実行モデルの構築，情報処理学会論文誌コンピューティングシステム， Vol.46, No.12, pp.181–193 (2005).