生命科学論文データベースの全データからのタンパク質に関する生命事象の抽出

田浦健次朗* 鴨志田良和† 横山大作‡ 頓楠 * 崔升準 *
柴田剛志§ 三輪誠 § 松崎拓也 * 辻井潤一 §

1 はじめに

近年の生命科学分野における研究の進展はめざましく，それに比例して毎年巨大な量の論文が生成され，いまもその量は増加し続けている．一方で，研究者が読むことができる論文の数は限られており，次々と報告される研究成果を広範囲に理解し，自らの持つ知識を常に最新の状態に更新し続けることは年々難しくなってきている．生命科学論文データベース Medline は，この分野における論文データベースとしてはもっとも巨大なデータベースの一つである．現在，Medline データベースには約 1800 万件の論文が登録されており，さらに一日あたり 1000 件以上の論文が新たに登録されている．Medline のようなデータベースから自動的に情報を抽出し，研究者が知りたい情報を簡単に得ることができるようなシステムの必要性が年々高まっている．このような情報の抽出・検索技術に対しては，抽出・検出したい情報が高度なものになればなるほど，そのために必要となる自然言語処理技術も高度なものになる．

タンパク質に関係する生命事象 (イベント) に関する情報は，そのような高度な情報の一つである．イベントには様々な種類があり，文章として表現されているそれらのイベントを正しく判断し，推定するということを自動で行わなければならない．そのようなことを行うために必要となる自然言語処理技術は，係り受けなどの浅い構文解析だけでなく，意味のレベルを表現するような構造を出力する深い構文解析や，書かれているタンパク質名や遺伝子名を表すフレーズを発見するタグ付けが必要となる．また，イベントは，普通に人間に読まれる文章として様々な表現で記述されるため，上記のような高度な自然言語処理技術の結果を利用して，イベントの有無および種類を推定する必要がある．この推定の仕方の詳細については次節で説明する．今回の実験では，以上のような処理を Medline に登録されている全論文のアブストラクトに対して行い，イベントを出力する．

このような計算は大規模計算の観点からみると，次にあげるような二つの特徴を持つ．一つ

* 東京大学大学院情報理工学系研究科
† 東京大学情報基盤センター
‡ 東京大学生産技術研究所
§ 東京大学情報学環

目の特徴は，重要な計算部分では，複数のアブストラクトをまたぐようなデータのやりとりが存在しない点である．このことは，アブストラクトの単位で分割し，独立して処理を行うことができることを意味する．これは，近年大規模データ処理にしばしば使われるシステムである MapReduce やワークフローソルバーのような，タスクレベルに分割するような計算モデルを用いることができる．これらのプログラムモデルの特徴は，並列計算に関わる様々な複雑な部分，例えば負荷分散，データの通信や同期，スレッド処理などの記述を必要とせず，また，したがって，並列計算を効率的におこなうためにローレベルなプログラムの工夫を必要としないという利点がある．なお，このような計算モデルが適さない例としては，例えば行列演算のようなタスクをあげることができる．行列計算では並列に演算しようとしても，各計算を独立して行うことができず密にデータ交換する必要がある．二つ目の特徴は，前述のように，様々な自然言語処理技術や機械学習の技術を寄せ集めて用いる必要があるという点である．これらの技術は，今回のタスクだけのために一から開発することは不可能に近く，様々な研究者によって作られ公開されたツールを使うことになる．この場合，これらのツールのソースコードを理解する必要はなく，どのような入力を与えればどのような出力がでるか，必要な計算時間とメモリはおおよそどれぐらいかがわかっていれば十分である．また，その際使われているプログラム言語もインタプリタ言語を含め様々である．ワークフローシステムは，このようなプログラムを組み合わせるのに最適な計算モデルである．今回の実験では，GXP make とよばれるワークフローシステムを用いて処理を行った．この処理に関する詳細は 3 節で説明する．

2 タンパク質に関するイベントの抽出

1 節で述べたように，生命科学文書からの高度な情報抽出の重要なトピックのひとつとして，文中に出現する生命科学用語の間の意味的な関係を抽出するというものがある．タンパク質間相互作用，遺伝子・病気の因果関係，タンパク質と種の表現 (human, mouse など) など文中に出てきた 2 つの生命科学用語の間に関係が存在するか，しないかというの 2 値の情報を抽出するという研究は長年行われてきたが，ここ数年，自然言語処理技術の発展につれて，単に 2 値では表現できない，より複雑な関係についての抽出手法に関しても，現実的で有用な方法が開発されてきている．BioNLP'09[*1]の共通タスクでは，タンパク質に関係する生命事象 (イベント) と呼ばれる，遺伝子やタンパク質間における複数の与えられた意味的関係を文中から抽出し，その精度が競われた．本稿では，BioNLP'09 で最も優秀であった手法よりもさらに高い精度が得られるイベント抽出システム [10, 11] を用いて抽出を行った．

2.1 抽出するイベント

まずはじめに，タンパク質に関するイベントとはどのようなものかについて例をとって見てみよう．図 1 に示すような文があったとする．この文を直接読むと，次のような複数のたんぱく質に関連する現象が記述されていることがわかるはずである．

[*1] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/SharedTask/

(1) $TRAF_2$ のリン酸化 (phosphorylation) が行われること．
(2) $TRAF_2$ が CD_{40} の細胞質ドメイン (cytoplasmic domain) に結合 (bind) すること．
(3) 上記 (1) の現象が原因で，上記 (2) の現象が抑制 (inhibit) されること．

それぞれ，(1) はリン酸化，(2) は結合，(3) は負の制御，というたんぱく質に関するイベントが起きていると分類される．また，それぞれのイベントは，主題や原因などをあらわすあらかじめ決められた項をとり，そこにはタンパク質名である TRF_2 や CD_{40} や，そのほかのイベント自体などが代入される．その結果，抽出すべき対象となるイベントは，図 1 下部に描かれているような構造となる．

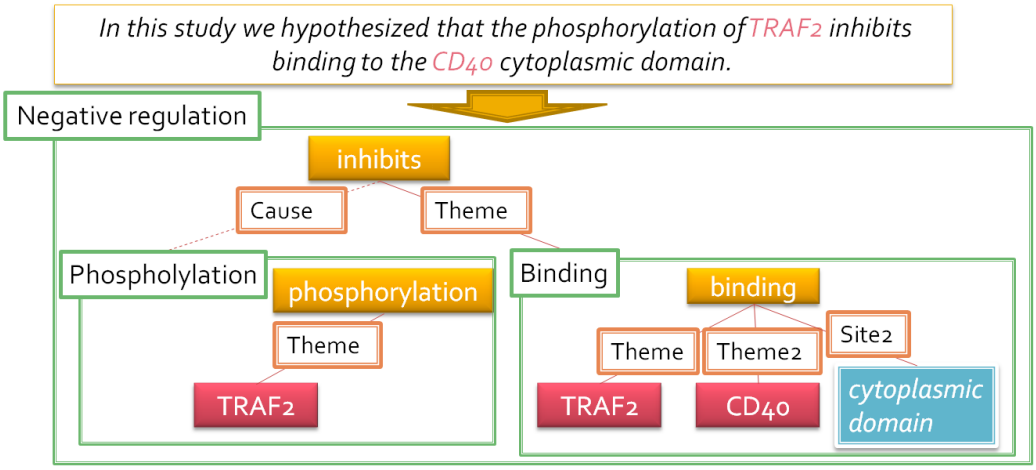

図 1　抽出されるべきイベントの例

さて，BioNLP'09 共通タスクでは，三つのサブタスク

(1) イベントの発見
(2) 場所やサイトを表す表現を第二の項として抽出
(3) イベントが推定や否定として述べられているとき，そのことの認識

をテーマとしてその精度が競われたが，本稿で用いる手法は上記のうち (1) と (2) のサブタスクを行うものである．イベントの種類は 9 個あり，それらは次の二つのカテゴリに分けられる．

(1) 単純なイベント：
・遺伝子の発現 (Gene Expression)・タンパク質の異化 (Protein Catabolism) ・転写 (Transcription) ・リン酸化 (Phosphorylation) ・局在化 (Localization)
(2) 複雑なイベント：
・結合 (Binding) ・制御 (Regulation) ・正の制御 (Positive Regulation) ・負の制御 (Negative Regulation)

単純なイベントでは，その項として主題を一つだけ取り，かつ主題はタンパク質 (または遺伝子や RNA) の名前でなければならない．一方，複雑なイベントでは，複数の主題が存在するか，主

題以外に原因を表す項が必要となる．また，主題・原因を表す項に，タンパク質でなくイベントを取ることもある．例えば，図 1 の例で示したように，複雑なイベント「負の制御」に対して，その「主題」の項として $TRAF_2$ が CD_{40} の細胞質ドメインと結合するというイベントが，その「原因」の項として $TRAF_2$ のリン酸化というイベントがとられている．

表 2.1 に上記で述べた各イベントおよびそのイベントが取るべき項を示す．図中 (G) は遺伝子およびその生成物，(L) は場所，(E) はイベントを項として取ることを意味している．また'+' は追加的にとる，という意味である．

表 1　各イベントの種類および項

イベントの種類	項
Gene Expression	Theme (G)
Protein Catabolism	Theme (G)
Transcription	Theme (G)
Phosphorylation	Theme (G) +Site (L)
Localization	Theme (G) +AtLoc (L) +ToLoc (L)
Binding	Theme (G) +$\text{Theme}_{2\cdots n}$ (G) +$\text{Site}_{1\cdots n}$ (L)
Positive Regulation	Theme (G/E) +Cause (G/E) +Site (L) +CSite (L)
Negative Regulation	Theme (G/E) +Cause (G/E) +Site (L) +CSite (L)
Regulation	Theme (G/E) +Cause (G/E) +Site (L) +CSite (L)

2.2　基本となる自然言語処理技術

上述のようなイベントを抽出するためには，まずはじめに，基本的な自然言語処理として，与えられた文に対して次のようなことを行う必要がある．

(1) 文を構成する単語に対して品詞タグ付けを行う．
(2) タンパク質や遺伝子の名前であるかどうかを判定する．
(3) 文の構文を解析する．

品詞のタグ付けを行うタスクは，生命科学分野固有のタスクというよりは，自然言語処理における一般的なタスクである．テキストのどの表現がタンパク質を表すのかは，タンパク質に様々な表記が存在し，簡単に特定することができない．例えば，'IL-2', 'IL2' や，'GHF-1 transcriptional factor', ' GHF-1 transcription factor ' などの表記の揺れがあり，また，同じ表記でもテキストの文脈によってタンパク質である場合とそうでない場合があるため，辞書を引くだけでは対応し切れない．今回のイベント抽出におけるタンパク質・遺伝子の発見では CRF (Conditional Random Fields) の手法を用いた固有表現抽出器を利用した．また，文の構文解析に関しては，古くから様々な研究がされており，単に構文と言っても，表面上の関係を木構造で表現する事ができるだけの浅いものから，単語間の意味上の関係までを記述できるような特殊な構造を出力の

構文とするものまでがある．Enju[*2]と呼ばれる構文解析器は後者の一例であり，その出力結果から，述語項構造と呼ばれる，各述語の意味上の主語や目的語を表す関係を取得することができる．イベント抽出タスクのために，上記の Enju の他に，GDep[*3] という構文解析器も用いており，その 2 つの構文解析の結果からの情報を特徴をとして用いている．GDep は Sagae らによって開発された依存解析器 (ksdep) を GENIA Treebank[*4] という医学文書に関するコーパスを用いて，確率的 shift-reduce アルゴリズムにおけるパラメータを学習させたものである [14, 12]．依存木 (dependency tree) とは，Penn Treebank[*5]のような句構造を持つ構造木とは異なり，単語間の依存関係によって表現される木である．たとえば，図 2 に示したような文に対して，単語間の依存関係が矢印で示すように与えられる．一方，深い構文解析 (Enju) では，図 3 に示すよう

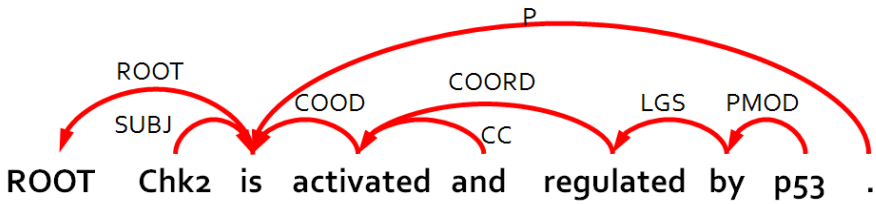

図 2　依存解析 (GDep) の出力結果

に，句構造 (NP, PP など) による構文木が得られるだけではなく，それ以外にも，意味のレベルまで踏み込んだ構造，たとえば述語項構造と呼ばれる構造も得られる．この述語項構造では，図中赤い線で示されているように，regulate の意味上の主語が by 以下の部分木で示される lysine demethylase および LSD1 であり，意味上の目的語が p53 であるといったことが抽出される．これは依存解析の結果では得ることができない．たとえば，述語項構造のエッジからなるグラフは木構造にはならない．

依存解析であれ深い構文解析であれ，構文解析の結果は，高度な情報抽出をするとき，たとえばタンパク質間の相互作用・イベントなどの情報を抽出するとき，機械学習のための有用な特徴量として利用することができる．たとえば，図 3 では，regulate の述語項構造における意味上の目的語が p53 となっているが，そのことが，lysine dethylase や LSD1 が p53 を制御するというイベントを機械学習によって発見するときに，特徴として加えられる．

また，一般的に，深い構文解析の結果は依存解析の結果より多くの情報を含んでいるが，タンパク質間の相互作用・イベントなどの高度な情報を抽出するときに機械学習器に与える特徴としては，ある構文解析の結果単体ではなく，複数の異なる解析器の結果を用いたほうがより精度が上がることが知られている [12, 10, 1].

[*2] http://www-tsujii.is.s.u-tokyo.ac.jp/enju/index.ja.html
[*3] http://people.ict.usc.edu/ sagae/parser/gdep/index.html
[*4] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/topics/Corpus/GTB.html
[*5] http://www.cis.upenn.edu/ treebank/

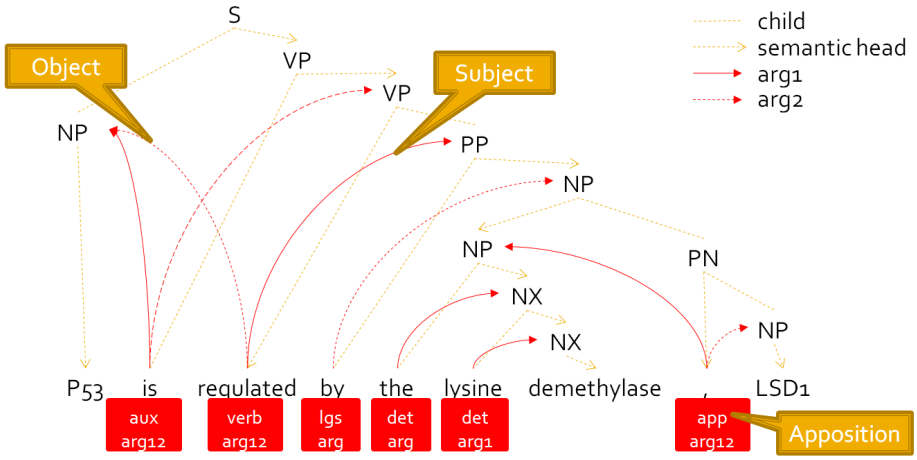

図 3　深い構文解析 (Enju) の出力結果

2.3　イベント認識システム

ここで紹介するシステムは，BioNLP'09 のイベント抽出タスクを 3 つのステージに分けて実行するものである．複数の高度な自然言語処理技術を部品として用いて機械学習のための特徴が作成されるため，全体として一つのプログラムではなく複数のプログラムから構成されており，計算時間も多く必要とするが，現在のところ他の手法に比べ最も高い精度が得ることができる．図 4 にイベント認識システムの概要を示す．図中，右の Event Extraction のところがイベント抽出のために新たに提案された手法である部分であり，Preprocess の部分は前節で述べたような既存の手法の組み合わせによって成り立っている．まず，与えられた論文データ (Annotated Corpus) から，文章部分を切りだし，それらを前節で説明したような，生命科学文書に特化した固有表現抽出，つまりタンパク質や遺伝子の名前かどうかの特定を行う (NER)．また，構文解析器である Enju と GDep によって各文の構文情報および意味情報を抽出する (Parsers)．それらの結果から，各文に対して，イベント抽出のための特徴を抽出する (Input for Detectors)．その特徴を利用して，3 段階にわけてイベント抽出を行う．各段階は，次のようになっている．

(1) イベントをあらわすトリガワード，および場所・部位をあらわす単語の判定.
(2) 各トリガワードに対して周辺の各表現がどの項になりえるかの判定.
(3) 複数項を取るような，複雑なイベントの判定.

なお，この三輪によるシステムでは全ての判定にサポートベクタマシン (SVM) という機械学習器を用いている．また，SVM は通常 2 値分類しかできないため，複数のクラスに分類する必要がある時は，1 対他の分類を SVM で行い，どのクラスに属するかを決定する手法を用いている．

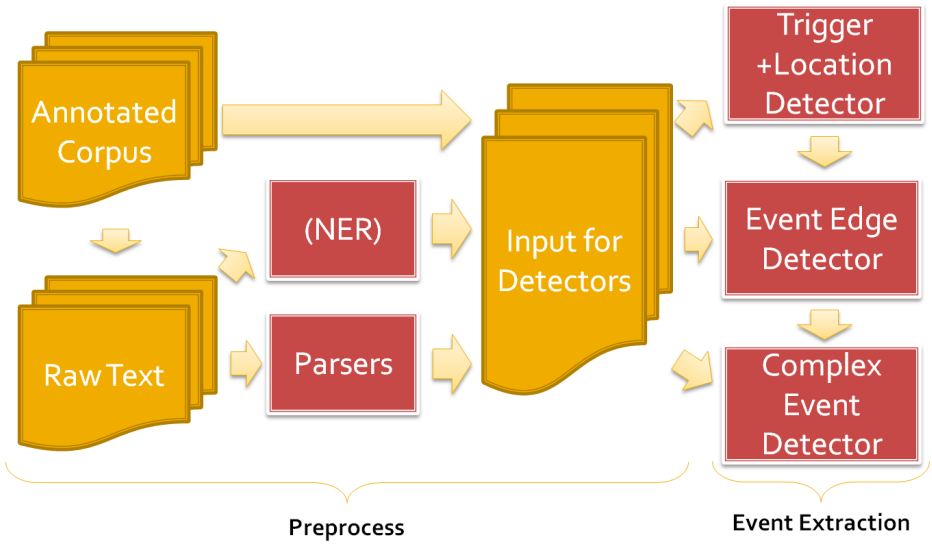

図 4　イベント認識システムの概要

2.3.1　ステップ 1：トリガワードおよび位置ワードの判定

まず，イベントが記述されているかどうかを判定しなければならないが，そのために，各イベントに対し，イベントのコアとなる単語を発見する．そのような単語はトリガワードと呼ばれる．おもにトリガワードを発見するがこのステップに相当する．それには，Enju や GDep, NER の結果を特徴として，各単語に対して，それがトリガワードになりえるかどうか，また成りえるとすればどのイベントのトリガワードになるかを判定する．たとえば，図 5 のような文に対しては，文中の the や of に対しては，それぞれに対する作成された特徴を入力として機械学習器（ここでは SVM）で判定するによって，トリガワードにはならない (Neg) と判定される．一方，phosphorylation や inhibits といった単語は，それぞれ，リン酸化や負の制御のイベントをあらわすトリガワードとして判定される．

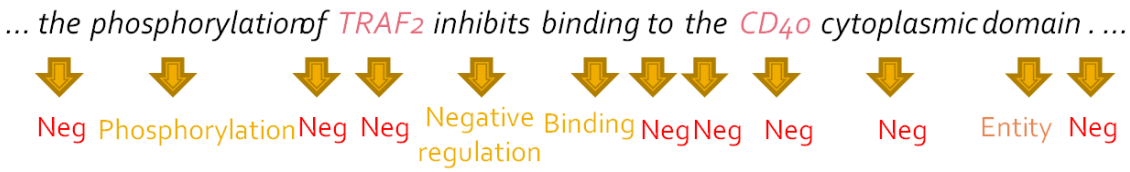

図 5　ステップ 1

2.3.2　ステップ 2：各イベントの項の特定

ステップ 1 で，トリガワードによって，イベントが存在することと，イベントの種類を判定した．次は判定されたイベントに対して，その項（表 2.1 における Theme など）を特定する必要がある．そのために，このステップではまず，すべての項となりうる表現とイベント（トリガワー

ド）の組み合わせをエッジとして列挙する．そして，そのすべてのエッジに対して，順に，その単語が項となりうるかを機械学習を用いて判定する．ここで，機械学習で利用する特徴は，ステップ 1 と同じように，Enju や GDep, NER の結果から作成されたものである．また，項には複数の種類あるので，ステップ 1 と同様，マルチクラス，マルチラベルの判定となる．たとえば，図 6 上のようにトリガワードが決定していた場合，各トリガワードである，phosphorylation, inhibits, binding に対する項としてとりえるものは，TRAF2, CD40, domain という単語である．トリガワードおよび特定されたイベント名と，それらの単語の組み合わせに対して，機械学習器を用いて判定した結果，たとえば，トリガワード phosphorylation に対してタンパク質名 TRAF2 はテーマという項であると判定される一方，同じ phosphorylation に対してタンパク質名 CD40 は項とはならない，と判定される．

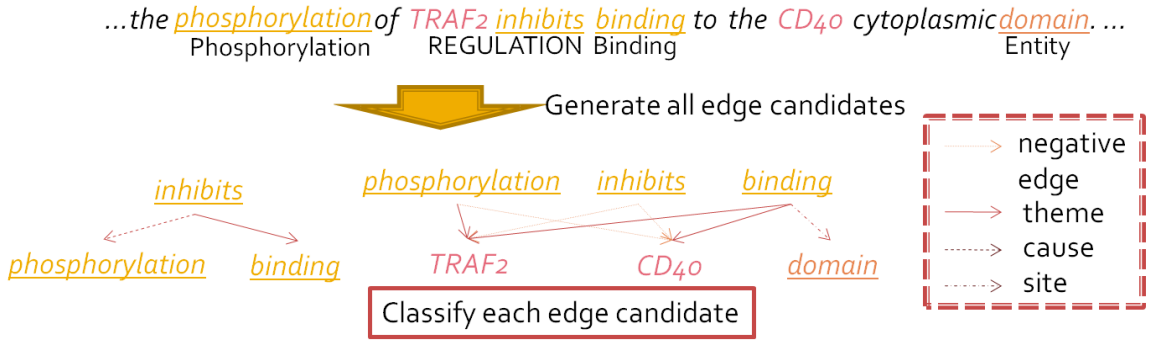

図 6　ステップ 2

2.3.3　ステップ 3：複雑なイベントの特定

遺伝子発現やリン酸化など表 2.1 の上から三つ目までのいわゆる単純なイベントでさらに項を一つしか持たないものに対しては，ステップ 2 ですべての項を推定し終わっている．もし，二つの単語が theme として判定されるなど，重複するような複数の項が得られたら，二つのイベントとして出力すればよい．一方，表 2.1 の四つ目以降のもの，つまり複数の項をもつ単純なイベントで項を複数もつ可能性のあるもの，または複雑なイベントに対しては，複数の項を取る可能性があるため，ステップ 2 の結果で十分に整合性があり合理的な結果が得られているとは限らない．そこで，複雑なイベントをあらわすトリガワードに対しては，ステップ 2 で項となると判断されたものすべてに対して，それらの複数の組み合わせを列挙する．そして列挙されたトリガワードと複数の項の組み合わせに対して，機械学習器を用いて，イベントとして正しい組み合わせかどうかを判定する．正しい組み合わせと二つ以上のものが判断されたら，機械学習器の出力する確度を比べて高いものを採用する．図 7 は図 6 の結果の続きであるが，inhibits, binding という二つの複雑なイベントをあらわすトリガワードが存在する．そして，図の左下に描かれているように，inhibits に対しては二つの項をとるか一つだけ項を取るかの二通りの場合があるが，そのうち二つの項を取るものが正しい組み合わせとしてこのステップの結果として判定されている．同じように，binding に対しては三つの項をとるものが正しい組み合わせとして判定されている．

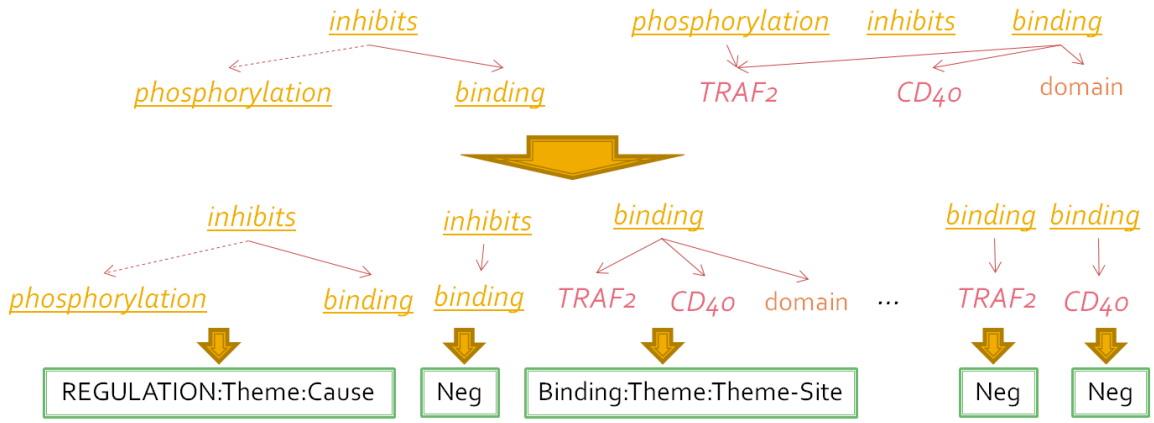

図 7　ステップ 3

以上の 3 つのステップの結果，得られた出力の例を図 8 に示す．表記の都合上，Localization は二つのイベントであったものを一つにまとめている．

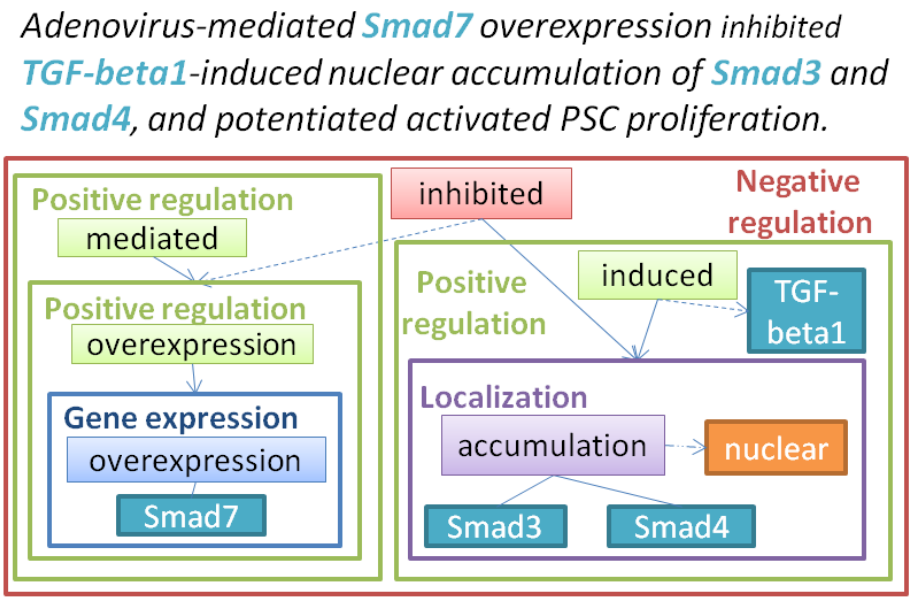

図 8　イベント認識システムによる出力の例

3　ワークフローシステムによる全データの処理

今回 T2K 上で行った実験では，GXP make という，著者らが開発・研究しているワークフローシステムを用いている．前述のような Medline 全データからのイベントの抽出以外に，今回の実験のもう一つの意義は，ワークフローシステムの大規模運用を実際に行なうことで，運用の

表 2　ワークフローシステムの例

	記述言語	部品	主な想定環境
Swift[17]	SwiftScript	executable	HPC
Dryad[8]	C++	executable	HPC
Xcrypt[6]	Perl dialect	executable	HPC
Taverna[13]	GUI	Web service	WWW
Triana[2]	GUI	Java class	LCS
Kepler[9]	GUI	Java class	LCS
Pwrake[15]	Rake	executable	HPC
makeflow[16]	make-like	executable	LCS
SGE qmake[5]	make	executable	HPC
DAGMan[3]	static DAG	executable	LCS
Pegasus[4]	static DAG	executable	LCS

実績や，大規模化による問題があるならば問題点を洗い出すことにもある．

3.1　ワークフローシステム

1 節でも述べた通り, 本タスクの処理は大量の文献データに対して様々な既存の自然言語処理プログラムを適用する事が中心であり, そのような処理には「ワークフローシステム」と呼ばれるシステムが適している．一般にワークフローシステムは各タスク (コマンド) のコマンドラインと, タスク間の依存関係を記述した「ワークフロー記述」—本質的にはタスクを頂点とした DAG—を受け取り, タスクが実行可能になった (依存関係で先行するタスクがすべて終了した) 時点で計算ノードへ自動的に負荷分散して実行する．数多くのワークフローシステムとして提案されており, 代表的なものを表 2 にあげた．

本課題では筆者が設計・開発している処理系である GXP make[*6]を用いている．本システムは,

- 名前から想像されるとおり, Unix の Makefile によってワークフローを記述する
- HA8000 システムはもちろんのこと, SSH, TORQUE, Sun Grid Engine などで管理された多種多様な環境, それらが混在した環境で動作する[*7]
- いったんバッチスケジューラ経由で計算ノードを取得した後は, ワークフロー中の個々のタスクはバッチスケジューラを介さずに投入されるため, スループットが高い

などの特徴を持つ．Makefile の読み込みからタスクの生成までは GNU make を修正せずに用いており, GNU make が持つすべての機能をサポートする．このことで, make が元々持っている様々な利点:

[*6] http://www.logos.ic.i.u-tokyo.ac.jp/gxp/

[*7] T2K の京大システム HX600 や, 東工大の TSUBAME での動作実績もある

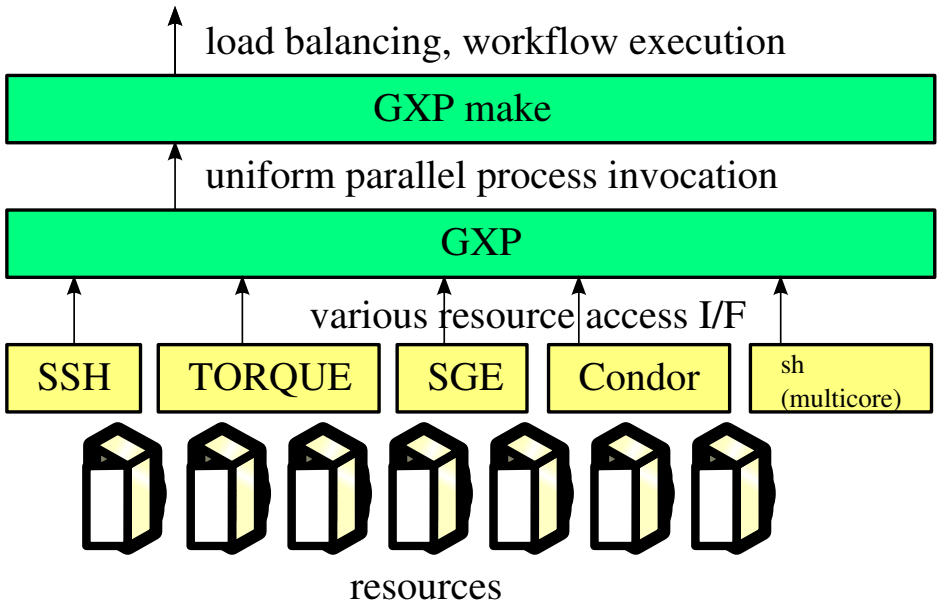

図 9　GXP 実装の概念図

- 多くの Unix ユーザにとって学習コストが非常に小さく
- ファイルの存在や更新日時に基づいた依存解析が行われるため, 途中で失敗したワークフローを続きからやり直す事が容易にできる (その意味での耐故障性がある)
- パターンルールやワイルドカード関数などの機能を使うことで, 多数のタスクを少ない記述で生成できる

などの利点がほぼ自動的に引き継がれている．特に, 1 ノード内での並列処理のために GNU make の機能 (-j) を用いていた処理は, ほとんど書き換えの必要なく, クラスタでの並列実行が可能になる.

図 9 に GXP make 実装の概念図を示す．一般に個々の計算資源は, あるものは SSH などの遠隔ログインシステム, あるものは TORQUE, SGE などの遠隔ログインシステムなどを経由してアクセスされる．言い換えれば計算資源は多様なインタフェースを持っている．また一般にスーパーコンピュータなどの大規模なクラスタはいったんログインノード (通常 SSH でアクセスされる) を介してでないとアクセスできない．それらの多様性を吸収して, 個々のコマンドを統一的な手段で実行できるようにするのが, GXP である．また, いったん SSH やキューイングコマンド (qsub など) によって資源を獲得したら, 個々のコマンドはメッセージを送るだけで軽量に行われる．GXP make はその統一的なコマンド起動の API を用いて, Makefile から生成された個々のジョブを実行する.

3.2　実験環境のセッティングおよび実行

本実験においては，HA8000 の 512 ノード, 8192 コアを用いた高い並列度での実行を行うことはもちろんであるが, 遠隔の拠点を連携させた実行を行うことも目指した．具体的には, 特段連携

に関する準備をしていない複数のクラスタを，実行途中に計算に参加させ，連携して利用することを試みた．これは以下のような様々な状況で必要になる．

- 複数のクラスタに利用権を持つ利用者が, 通常は小規模なクラスタで計算を行い, 大規模クラスタを利用可能な時間だけ参加させる．
- ストレージ (データ) と計算資源が遠隔あるいは異なる管理ドメインにある．

実験に用いた計算機環境を以下に示す．

コントロールノード　InTrigger クラスタ[*8]の 1 ノード (hongo) を利用．設置場所は東大工学部 2 号館．
Intel Xeon X5560, 2.8GHz, 8 core. 24GB memory, 10GBase-ether.

HA8000　AMD Opteron 8356, 2.3GHz, 16 core. 32GB memory, Myrinet-10G. 512 nodes (total 8192 cores)

生産研クラスタ (cloko)　東京大学生産技術研究所に設置したクラスタを利用．
Intel Xeon E5530, 2.4GHz, 8 core. 24GB memory, 10GBase-ether. 60 nodes (total 480 cores).

また，処理したデータは，medline10n0300 から medline10n0623 の 324 個の PubMed のデータセットである．これはデータのサイズで考えると，データセット全体の大半を占める部分であり，処理すべき文が約 7200 万個 (単語の数にすると約 15 億個) 含まれている．

GXP make は，1 台のコントロールノードが全てのワーカの実行を制御する，マスタワーカ方式の構造を持つ．コントロールノードでは，UNIX make を用いた未実行タスクの発見とスケジューリング，ワーカの状態管理とアイドルなワーカへのタスクディスパッチなどを行う．ワーカノードは指示されたタスクを実行するだけの制御構造である．本実験では，ワーカノードとして HA8000 と生産研クラスタの 2 クラスタを用いた．

3.2.1 同一ファイルシステムイメージの実現

GXP make は，ワークフローのスケジューリングのために UNIX の make を用いる．中間タスクの出力ファイルと依存関係グラフを元に実行すべきタスクを決定しているのは，コントロールノードで実行される make プロセスであるため，コントロールノードから，全てのタスクの出力が同一ファイルシステム上のファイルとしてアクセスできなければならない．また，ワーカ間の中間ファイルの受け渡しについても，ステージング等の通信は行わないため，ワーカ群についても同一ファイルシステムイメージを共有する必要がある．

本実験では，コントロールノード，HA8000, 生産研クラスタの管理の異なる 3 つのクラスタを連携させることを目指す．連携に際しては，システムソフトウェアや管理体制に関する制約をできるだけ少なくすることが望ましい．そのため本実験では，ユーザ空間ファイルシステムを実現する FUSE と SSH を用いる，SSHFS を利用することとした．FUSE は最近の Linux カーネル

[*8] http://www.intrigger.jp/

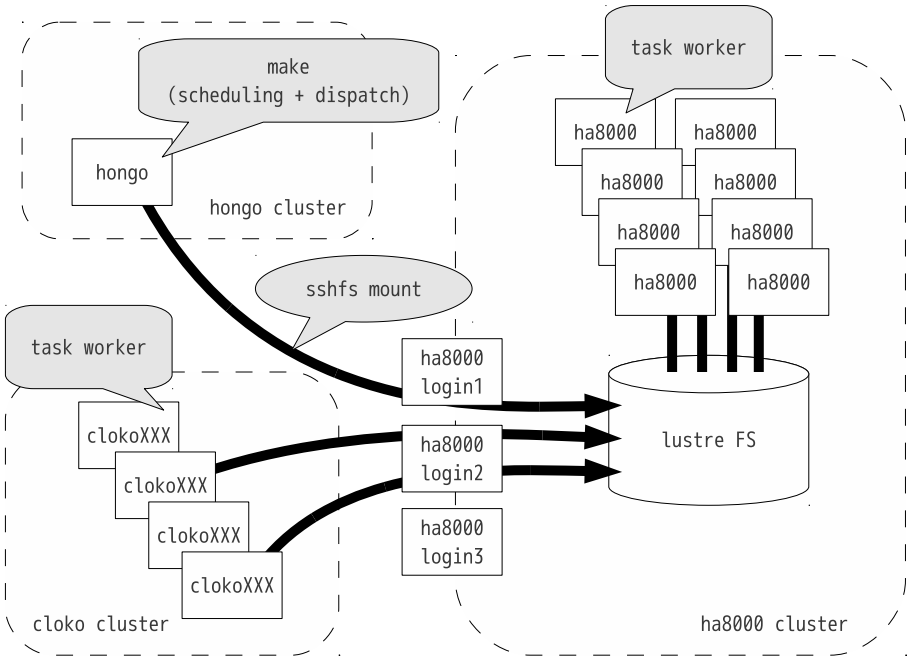

図 10　計算に利用した機器構成

では標準的に利用できる機能であり，SSH と組み合わせた SSHFS を用いることで，ユーザがログインできる任意のノードのファイルシステムを，手元のノードにユーザ権限でマウントし，手元のファイルシステムの一部として透過的に取り扱うことが可能になる．

ここでは，ほとんどの計算が HA8000 クラスタで実行されることを考慮し，実際にデータが置かれる場所は HA8000 の Lustre ファイルシステムとした．コントロールノード (hongo) 並びに生産研クラスタ (cloko) は，HA8000 のファイルシステムを SSHFS でマウントして用いた (図 10)．hongo と cloko から HA8000 への SSH アクセスは，HA8000 のログインノードを通してしか行えないため，SSHFS のマウントではログインノード 3 ノードをラウンドロビンで利用し，通信負荷の均衡を図った．

3.2.2　計算中の構成変更

実験においては，

- 最初に，HA8000 クラスタ 512 ノードによる計算実行を開始する．
- HA8000 の利用可能時間が残り 40 分程度になったところで，別のクラスタ (cloko)60 ノードを計算に参加させ，HA8000 にはそれ以降タスクをスケジューリングしないようにする．一時的に HA8000 と cloko が混在した構成で並列実行が行われるが，割り当て済みのタスクが終了し次第，HA8000 から cloko へタスク実行が移行することになる．
- HA8000 の利用時間が終了した時点で，HA8000 の計算リソースを解放，以後は cloko のみで残りのタスクの計算を行う．

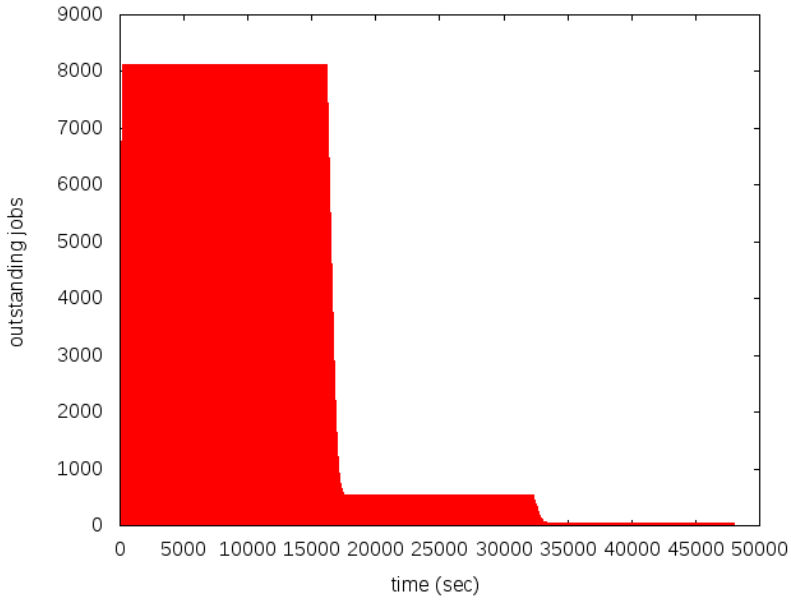

図 11　並列度推移

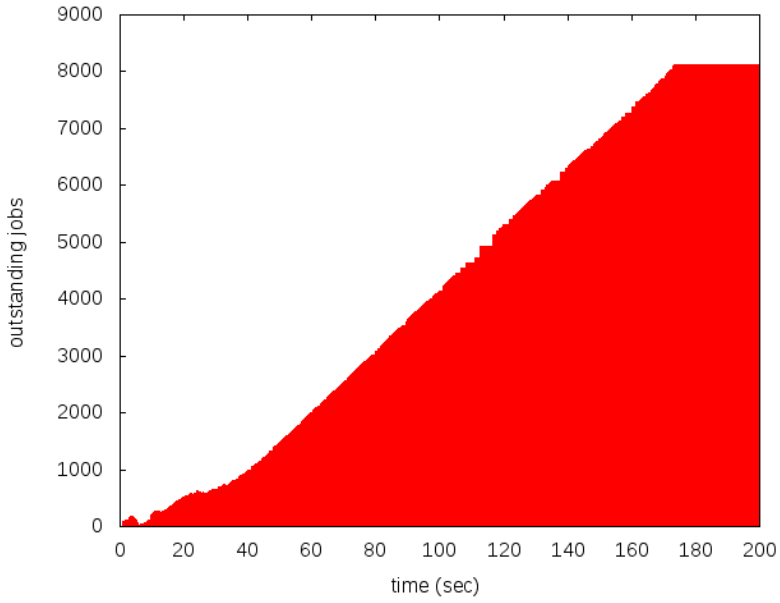

図 12　並列度推移 (計算開始時近傍を拡大)

というシナリオで計算ノードの参加・脱退を行った.

今回の実験におけるタスクの並列度推移を図 11 に示す．横軸は計算開始時からの経過時間 (秒)，縦軸はその時点で並列実行されているタスク数を表す．開始時点より速やかに HA8000 クラスタ全てを利用した 8000 並列での実行が行われている (図 12 参照)．4.5 時間 (16000 秒) 程度経過した時点で cloko クラスタのノードを追加し，HA8000 への新規タスク割り当てを止めたため，並列度は順次低下し，18000 秒程度経過した時点で HA8000 のタスクはほぼ終了し，cloko

のみが約 500 並列で計算を継続していることが見て取れる．

結果として，処理対象として想定した 7200 万文のデータセットについて，HA8000 と別のクラスタを組み合わせて使うことで，35000 秒でほぼ全てのタスクを終えることができた．

この実験はまた，大規模クラスタである HA8000 を計算リソースとして用いるだけでなく，大規模高速ストレージとして他クラスタから利用する試みでもあった．ユーザ権限のみで，一時的に共有ファイルシステムを構築し，本アプリケーションを実行するために十分な性能で利用することが可能であると示すことができた．

3.3 InTrigger 環境における実行

前述のように HA8000 上でイベント抽出ワークフローに対して全てのデータの大部分を処理したが，残りは InTrigger [7] プラットフォームにおいて実行した．InTrigger において処理した分は，データセットのうち，medline10n0001 から medline10n0299 の 299 個と，medline10n0624 から medline10n0659 の 36 個である[*9]．この実験には，実際にデータを処理する以外に，地理的に離れた，従って遅延の高い，十数個のクラスタで，ワークフローシステム GXP make がどのようにふるまうかやどの程度効率的に実行することができるかを観測するという意味もある．

一般的にいって，ワークフローシステムでは中間データやインプットデータを全ての計算ノードから発見したり読み書きができるようにしておく必要がある．つまり，全ての計算ノードにまたがった共有ファイルシステムが必要となる．HA8000 上では，Lustre[*10]がその役目を果たした．しかし，複数クラスタ上では，NFS や Lustre のような実績のある標準的な共有ファイルシステムを使うことができない．それらは基本的に一拠点内に存在するクラスタまたはスーパーコンピュータを想定しており，複数拠点にまたがり，その間を広域ネットワークで結んでいるような環境は想定されていない．実際，広域ネットワークを用いると，NAT やファイアーウォールが存在するため，拠点間で自由に接続することができないことがあたるため，動かすことは難しい．また，もし仮に動くようにできたとしても，レイテンシがクラスタ内と比べて飛躍的に（1000 倍以上）大きくなるため，その影響を無視することができなくなり，それらの共有ファイルシステムでは効率的に動かない可能性もある．

そこで今回は，一つのノード上からアクセス可能なファイルシステムのディレクトリを，全ての計算ノード上の特定のファイルシステムの特定のディレクトリにマウントすることによって，ファイルシステムの共有を実現した．リモートにあるファイルシステムのディレクトリをローカルのファイルシステムのディレクトリにマウントするためのネットワークファイルシステムとして，Dun らが開発した SSHFS-MUX[*11]というツールを用いた．SSHFS-MUX は，既存のネットワークファイルシステムである SSHFS[*12]に改良を加えたものであり，元の SSHFS よりも高スループットが期待できる．なお，SSHFS は，SFTP のプロトコルを用いてネットワークファ

[*9] 初めの 299 個は数だけ見ると半分近くに見えるが，古い時期の生物医学論文であり，1 つのデータセットあたりの論文数が少なく，後半部分と比べ負荷が少ない．

[*10] http://wiki.lustre.org/

[*11] http://web.yl.is.s.u-tokyo.ac.jp/ dunnan/SSHFS-mux/

[*12] http://fuse.sourceforge.net/SSHFS.html

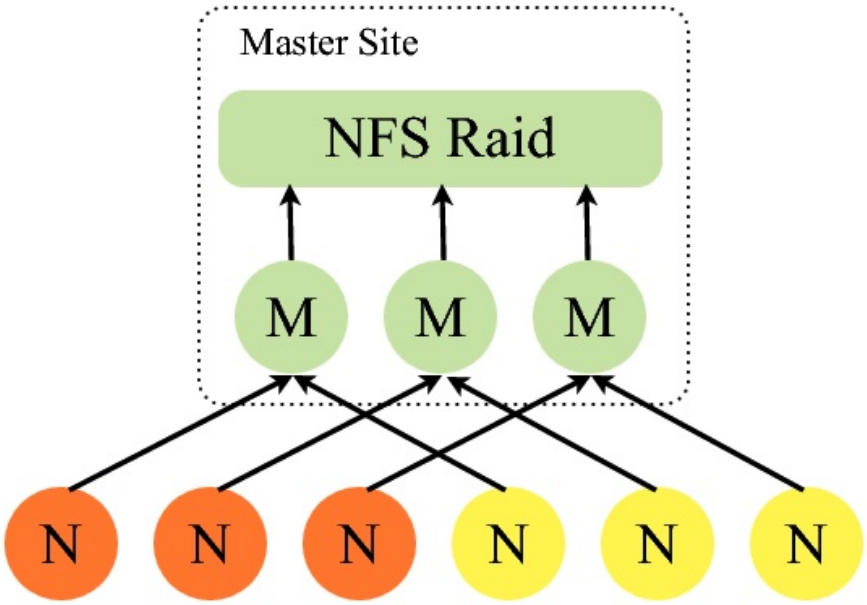

図 13　Mount Scheme

イルシステムを実現するものであり，FUSE[*13]というカーネルモジュールを利用することよってユーザレベルのプログラムとして動く．

ファイルシステムの共有の仕方は，全体として図 13 のように構成する．まずマスターとなるクラスタを一つ選び，そのクラスタ内の全てのノード (図中 M，以降 M と表記する) において，NFS Raid とよばれるレイドディスクをローカルにもつファイルシステムノードに対して，SSHFS-MUX でマウントする．さらに，他の拠点にあるクラスタのノード（図中 N，以降 N と表記する）からは，SSHFS-MUX で，ラウンドロビンに M を一つ選んで，それに対してマウントする．全体として，SSHFS-MUX によって，2 段のツリー状にマウントするような構造となる．ノード M は，結果として，ノード N と NFS Raid の間における I/O バッファーのような役割をはたす．

3.3.1　実験結果および考察

実験では，InTrigger の 11 拠点からそのとき利用できた合計 147 ノードをもちいた．処理するデータとしては，HA8000 で処理し残した部分，つまり medline10n0624 から medline10n0659 の 36 個の PubMed のデータセットであり，5732 個の XML ファイルを含む．各ノードに対して最大 4 の並列度を割り当てたため，並列度は全部で 588 となった．並列度としては HA8000 での実験における並列度に比べて 1/10 程度である．

ワークフローは 14763.7 秒で終了した．総ジョブ数は 11594 であった．全てのジョブのうち，簡単なジョブをのぞいた，主要なジョブ（イベント認識に関するジョブ）は 5732 個あり，それら

[*13] http://fuse.sourceforge.net/

の実行時間の平均は 1296.7 秒だった．また，標準偏差は 695.9 秒だった．

図 14 は実験のワークフローの実行における並列度の推移を表している．図を見てわかるように，最大の並列度である 588 並列に，ワークフローの開始後すぐに達している事がわかる．

1 クラスタ内での実行において終了直前まで並列度が高いまま維持できるようなワークフローの場合でも，複数クラスタ環境においては，通常，ロングテール，すなわち並列度が徐々に下がり最終的に並列度が 0 になるまでの間，ワークフローの全ての処理が終わるまでの”尾”は長くなる傾向がある．これは，複数の拠点を用いると，バンド幅が小さかったり遅延が大きかったりするなどして I/O のスループットが他と比べて突出して低い拠点が存在することが多いため，その拠点の計算ノードに割り当てられたジョブの実行時間が非常に遅くなるためである．ワークフローをロングテールとならないように賢く実行させるためには，このような状況を何らかの方法でモニターして把握し，スループットが非常に低いサイトにはあまり割り当てないような，ジョブスケジューリングをする必要がある．

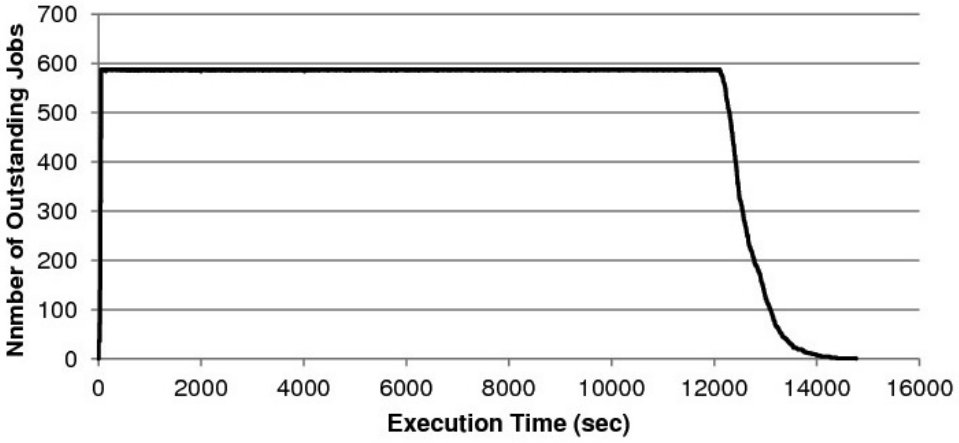

図 14　Parallelism during workflow execution on InTrigger

図 15 に示すようように，GXP make のマスターノード（すなわち計算ノードにジョブを投げるノード）のロードアベレージはワークフローの開始直後の部分で非常に高い．本ワークフローでは，与えられた入力ファイルは，一番初めに多数の細かいサイズのファイルに分割され，その各分割ファイルに対して並列にジョブが実行される．ワークフローの開始直後は，各計算ノードに 4 つづつあるスロットがほぼ全てフリーになっているために，それらに一度にジョブが割り当てられる．一方，開始直後の時期が過ぎると，ほぼ全てのスロットにおいてジョブが開始され，マスターノードはそれらをまって，ジョブが終了したスロットに順に新たなジョブを割り当てていくようになる．そのため開始直後は他の期間と比較してロードアベレージが非常に高くなる．

図 16 は，並列 make から GXP make が受け取ったジョブ，GXP make が各スロットに投げたジョブ，および終了したジョブのそれぞれの累積を表している．

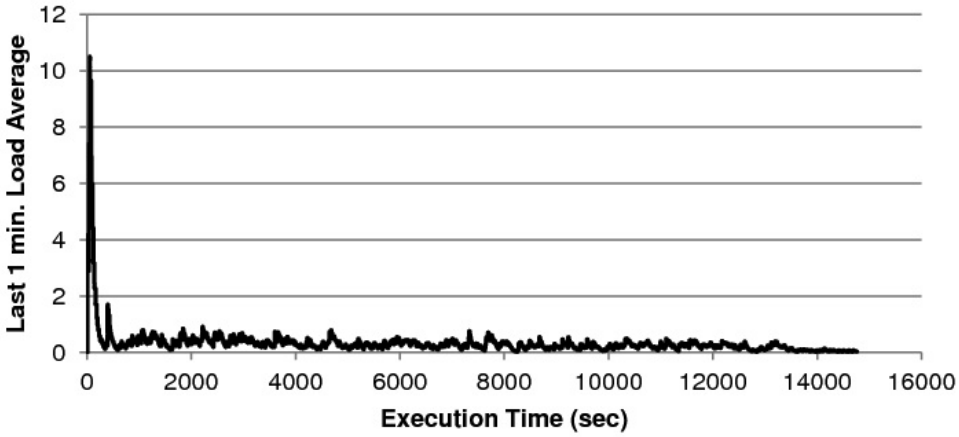

図 15　Load of dispatcher during workflow execution on InTrigger

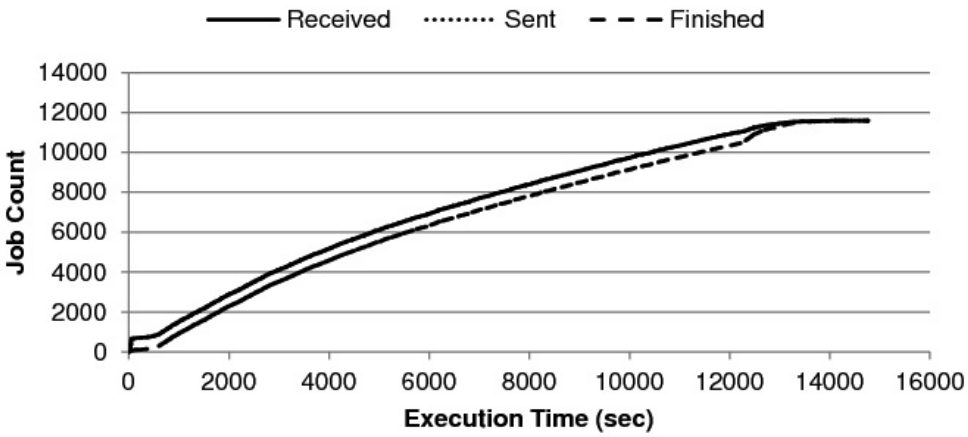

図 16　Jobs count during workflow execution on InTrigger

4　実験の結果：タンパク質間イベント情報グラフ

実行結果を解析したところ，表 3 に示したように 750 万程度のイベントを抽出できたことがわかった．これらのイベントから 150 のイベントをランダムに選び，人手での評価を行った結果，64% の正解率 (適合率) が得られた．今回処理したテキスト全体にどの程度のイベントが含まれているかは不明であるため，再現率の評価は難しいが，およそ 500 万弱のイベントが正しく抽出できているといえる．また，タンパク質 “IL-4” に関するイベントで頻度の多いものを実行結果から抽出した結果を図 17 に示した．この例では，タンパク質名の書かれたノードが (B)inding, (E)xpression, (P)ositive_regulation, (R)egualtion, (L)ocalization, (T)ranscription というイベントを表す表現を通じて接続されている．ただし，この図では，タンパク質を表すノードについては表現が同じであれば同じものとして描いており，またイベントについては表現と接続してい

るタンパク質が同じであれば同じとしている．どのタンパク質やイベントを同一視すべきかどうかについては，別の解決すべき問題として残されている．

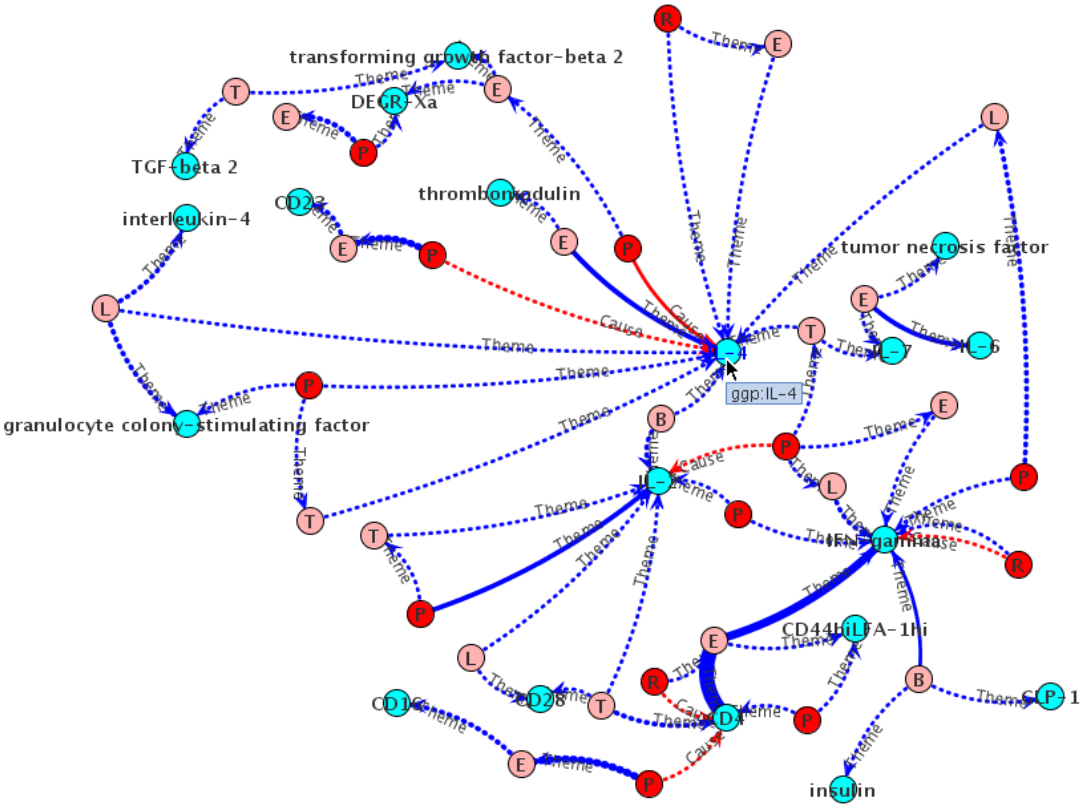

図 17　タンパク質 “IL-4” 周辺のイベント

今回の結果においては，64% という比較的高い適合率でイベントとテキストの対応がとれており，情報検索のインデックス・タンパク質代謝経路 (パスウェイ) の構築・表示における補助として利用できる．また，この処理により，テキストからイベントという抽象化された表現への対応が可能となり，またこの結果は複数の文もしくは文書に分散して記述してあるイベントをまとめて扱うための基盤として重要である．例えば，テキストではいくつかのイベントをまとめて 1 つのイベントとして (その間のイベントを省略して) 記述してあることもあり，その記載されているイベントの抽象度が様々であるが，このグラフを利用する (並行するパスを利用する) ことで，より利用者に見合った抽象度のイベントを生成・選択することが可能となる．

今後の課題としては，タンパク質の正規化を行う必要がある．正規化とは表現をタンパク質固有の (辞書) 識別子と対応させることであり，同じ表現でも異なるタンパク質を指す曖昧性・同じタンパク質でも異なる表現で表す表記揺れの 2 つの問題を解決できる．この正規化により，より柔軟な検索，より正確なグラフの作成が可能になる．

表 3　イベントの分布

Type	Count
Binding	906,904
Gene_expression	2,397,027
Localization	462,300
Negative_regulation	918,514
Phosphorylation	197,423
Positive_regulation	1,720,686
Protein_catabolism	64,661
Regulation	547,052
Transcription	323,843
ALL	7,538,410

5　おわりに

本稿では，HA8000 と外部クラスタとを連携して用いることにより，MEDLINE 全データベース約 7200 万文を使ってタンパク質に関わるイベントの抽出を行った．2 節で述べたように，用いた抽出システムは自然言語処理技術や機械学習手法のコンポーネントを多数寄せ集めたものとなっており，並列処理を行うためにはワークフローの枠組みで処理することが適当である．GXP make は著者らが開発しているワークフロー処理システムであり，3.1 節で述べたように，makefile の形式でワークフローを記述する．実験に用いたイベント抽出システムもワークフローとして makefile の形で書かれている．

本実験は全データを 2 回に分けて行われた．全体の約 9 割は HA8000 と外部にある東京大学生産技術研究所のクラスタ (cloko) とで連携して行われ，それらの二つの間は，外部ネットワークを通じて SSHFS を用いて入出力および中間ファイルの共有が行われた．正確には，3.2 節で述べたとおり，ほとんどの計算は HA8000 で行われており，HA8000 で割り当て時間を使い果たしたのちに実行を最後まで完了させるために cloko が用いられた．データは全て HA8000 のファイルシステム (Lustre) 上に置かれ，cloko クラスタを含め全ての計算機で共有されているため，割り当て時間を使い果たした後も，HA8000 がデータセンタのように機能したことになる．このような連携を用いない通常の利用方法，つまり HA8000 のみで計算を実行する場合では，ワークフローが途中で終了してしまうことを避けるため，確実に実行できる量しか投入できない．どのようなワークフローを実行するかにもよるが，完全に実行時間を予測することは困難なので，どうしても最後に計算をしていない空白時間が存在することになる．本実験では，HA8000 と外部のクラスタを連携させることによって，HA8000 の持ち時間終了時にワークフローが完全に終わる必要がなくなったので，図 11 からもわかるように，割り当て時間を最大限利用できた．

本実験によって得られた結果のデータは，過去の論文全データに対する結果であり，ME-

DIE[*14]などの高度な検索システムのデータベースとして組み込むことができる．また，今回の結果を踏まえて，抽出されたイベントから，タンパク質の正規化を行ったり，頻度が低く信頼度の低いエッジを除くなどの処理を行うことでより精度を高めたりすることで，より標準のパスウェイに近い形になおせば，現在手作業で行われているパスウェイのデータベースの作成などを半自動化することができるようになると考えている．

謝辞

本研究の一部は，文部科学省科学研究費補助金特別推進研究「高度言語理解のための意味・知識処理の基盤技術に関する研究」の助成を受けたものである．

参考文献

[1] Ekaterina Buyko and Udo Hahn. Evaluating the impact of alternative dependency graph encodings on solving event extraction tasks. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 982–992, Cambridge, MA, October 2010. Association for Computational Linguistics.

[2] David Churches, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang. Programming scientific and distributed workflow with Triana services. *Concurrency and Computation: Practice and Experience*, 18(10):1021–1037, August 2006.

[3] Peter Couvares, Tevik Kosar, Alain Roy, Jeff Weber, and Kent Wenger. *Workflows for e-Science: Scientific Workflows for Grids*, chapter Workflow in Condor. Springer Press, 2007.

[4] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[5] Grid Engine project home. http://gridengine.sunsource.net/.

[6] Tasuku Hiraishi, Tatsuya Abe, Yohei Miyake, Takeshi Iwashita, and Hiroshi Nakashima. Xcrypt: Flexible and intuitive job-parallel script. In *SACSIS*, pages 183–191, 2010. (in Japanese).

[7] InTrigger platform. http://www.intrigger.jp/.

[8] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, , and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[*14] http://www-tsujii.is.s.u-tokyo.ac.jp/medie/

[9] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, August 2006.

[10] Makoto Miwa, Sampo Pyysalo, Tadayoshi Hara, and Jun'ichi Tsujii. Evaluating dependency representations for event extraction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 779–787, Beijing, China, August 2010. Coling 2010 Organizing Committee.

[11] Makoto Miwa, Rune Saetre, Jin-Dong Kim, and Jun'ichi Tsujii. Event extraction with complex event classification using rich features. *Journal of Bioinformatics and Computational Biology*, 8(1):121–146, 2010.

[12] Y. Miyao, R. Saetre, K. Sagae, T. Matsuzaki, and J. Tsujii. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of the 45th Meeting of the Association for Computational Linguistics (ACL'08:HLT)*, pages 46–54, 2008.

[13] Thomas Oinn, Mark Greenwood, Matthew Addis, Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew Pocock, Martin Senger, Robert Stevens, Anil Wipat, and Christopher Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, August 2006.

[14] K. Sagae and J. Tsujii. Dependency parsing and domain adaptation with lr models and parser ensembles. In *proceedings of the CoNLL 2007 Shared Task. Joint Conferences on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*, pages 1044–1050, 2007.

[15] Masahiro Tanaka and Osamu Tatebe. Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing (poster). In *HPDC*, 2010.

[16] Li Yi, Christopher Moretti, Scott Emrich, Judd Kenneth, and Douglas Thain. Harnessing parallelism in multicore clusters with the all-pairs and wavefront abstractions. In *HPDC*, pages 1–10, 2009.

[17] Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor von Laszewski, Ioan Raicu, Tiberiu Stef-Praun, and Mike Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE International Workshop on Scientific Workflows*, 2007.