B-016

プル型ジョブスケジューラにおける動的通信量制御方式
Adaptive Communication Traffic Control Method for Pull-type Job Scheduler

齊藤隆之 † 善甫康成 ‡
Takayuki Saito Yasunari Zempo

1. はじめに

筆者らは，HPC 計算機クラスターの計算資源管理ミドルウェア ShareTask を開発してきた．[1, 2, 3] ラックに密に収容された計算機クラスターから，インターネット上に分散して存在する計算機クラスター群までを統一的に扱うことを目的にしているために，通信プロトコルとしては HTTP を使用している．

さらに，各ノードから管理サーバー（以下，ハブと呼ぶ）に対してのみ HTTP 接続を行う プル型方式 の採用により，計算機（以下，計算ノードあるいはノードと呼ぶ）あるいはネットワークの障害に対する耐性と，インターネット上の安全性への配慮を実現している．

ShareTask は，企業，大学，研究機関の HPC 計算資源を統合管理し，研究開発のためのさまざまな計算ジョブをスケジュールするために実際に利用されている．そこでは，高まる計算需要を賄うために計算ノード数が増強され続けており，ハードウェア障害への耐性（計算サービスの無停止性）が強く求められるため，ShareTask のプル型にもとづく耐障害性，特に障害箇所を自動的に切り離してシステム全体の計算サービスを無停止で継続する機能が高く評価されている．

しかし，プル型方式は，ハブ側イベントの計算ノードへの伝達遅延と，HTTP 接続の切断／再接続によるハブ側の負荷増大の問題を抱えており，計算ノード数が増大するに従い，伝達遅延と負荷増大が深刻な問題となる．ShareTask が応用が進み，制御する計算ノードの台数が増えるにしたがい，この問題解決が喫緊の課題となってきている．

Web アプリケーションの分野では，HTTP サーバーが同時に賄うクライアントの台数を増加させるために，シングルスレッドモデルと非同期 I/O モデルが注目され，JavaScript で記述された node.js [4] などの新しい HTTP サーバーが実用段階に入り，HTTP サーバーが同時に収容できるクライアント数が 10^3 から 10^4 の領域に至っている．また，伝達遅延問題の解決策として WebSocket など HTTP 上に双方向通信（bidirectional）のプロトコルを策定する動きも活発である．[5, 6]

ShareTask は，Web アプリケーションをベースとした計算資源管理ミドルウェアであるため，これら新しいしくみを容易に取り込むことが可能である．しかし，計算資源管理という観点から，HTTP の欠点を補うための従来手法もあわせて吟味し，総合的な解決手法を確立する必要がある．

本稿では，プル型方式における，計算資源制御の応答性能向上と，HTTP 通信負荷軽減の 2 つの課題を解決するための手法について検討する．

†(株) アンクル, ANCL, Inc.
‡法政大学 情報科学部

2.ShareTask の機能と構成

ShareTask は，以下の 3 種類の機能を統合したものである．

- 計算資源の利用分析と可視化（resource analysis）
- 計算資源のコントロール（resource control）
- ジョブスケジューリング（job scheduling）

多数のノードからなる分散システムにおいて，ハードウェア障害と通信障害も考慮しながら，各ノードとシステム全体を制御してこれら 3 種類の機能を実現するために，自律分散制御の考え方を採用している．ノードに常駐する制御プログラム（以下，エージェントと呼ぶ）が，ハブにノードの状態を通知するとともに，ハブから命令あるいは計算ジョブを取得してこれを処理する．(図 1) したがって，ハブは，各ノードの状態の詳細を知る必要はなく，ノードは随時，ハブから切り離すことが可能であり，ハブから切り離された状態でもノードは機能することができる．ハブとエージェントについて以下説明する．

ハブ　HTTP サーバ (Apache)，CGI 群 (Perl)，ならびにデータベース (PostgreSQL) とから構成された Web アプリケーションである．ノードの管理情報，ユーザ認証，各種ログ，計算ジョブの待ち行列を保持し，Web ブラウザとエージェントからの CGI 呼び出しに応答する．Web アプリケーションであるため，Web ブラウザからさまざまな操作ができる．エージェントからの報告によって収集された資源の状態は，データベースに蓄積，分析され，Web 画面で可視化される．プッシュ型との区別を明確にするために，中央管理サーバーあるいはマネージャーなどの呼称を使用せずに，エージェントが情報を交換する場所という意味でハブ（Hub）と呼ぶ．

エージェント　ノードに常駐して，ノードの内部状態を監視・制御するプログラムであり，ハブに対する HTTP クライアントである．ノードの OS 環境は多様であるために，エージェントはマルチプラットフォーム性が高い Java で開発されている．エージェントは，自律的かつ能動的に機能してノード内資源 (CPU，メモリ，HDD，プロセス) を管理する．ハブの CGI を呼び出すことにより，資源状態を報告するとともに，命令を受け取りそれを実行する．具体的には，以下の制御がある．

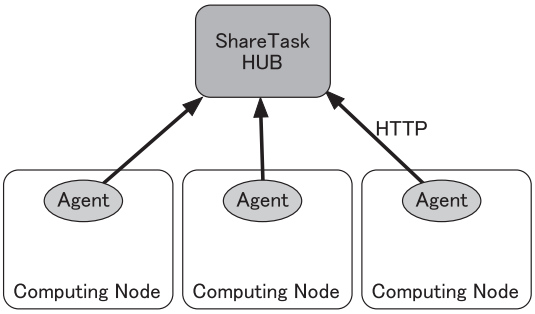

図 1: ShareTask の構成

- CPU，メモリ，HDD，プロセスを監視し，その使用状況をハブに報告する
- 計算ジョブの割当，実行，監視，一時停止，強制終了
- シャットダウン等 OS 全体に関わる制御

ノード・ハブ間の通信はつぎのように分類できる．

- イベント通知
- イベント取得（問い合わせ）
- ファイル転送（計算ジョブの入出力ファイル等）

イベントには，以下の種類がある．

- 状態（資源量，状態遷移の発生，実行待ちジョブの存在など）
- 命令（ジョブ強制終了等，ノード内部状態を変更する命令）

3. プル型の利点と課題

ShareTask では，各ノードは，そこに常駐するエージェントによって自律的に制御されていて，エージェントが自発的にハブに対して通信接続を行う．通信接続の方向性がノードからハブに向けてである．このため，ハブからノードへ向けての通信接続ができない環境，たとえば，NATP(IP マスカレード) の環境下にノードが配置されている状況においても，ノードの制御が可能である．

プッシュ型では，ノード数が増加するにしたがい，ハブからノードに対するポーリングのスケジュール (障害等でダウンしているノードの除外処理など) が煩雑になるが，プル型では，機能しているノードだけがハブに対して通信接続を行うので，ハブ側のスケジューリングを簡素化できる．

さらに，セキュリティの観点からは，ノードが外部からの通信接続を許可する必要がないので好ましいと言える．

一方，プル型方式が抱える本質的な問題は，イベント伝達が遅延する問題と，イベントを取得するためのポーリングにおいて無駄が多いことである．

3.1. イベント伝達遅延

ノード内で発生したイベントは，そのイベントに駆動されることでハブへの通信が開始される．その一方，ハブ側で発生したイベントについては，ノードが通信

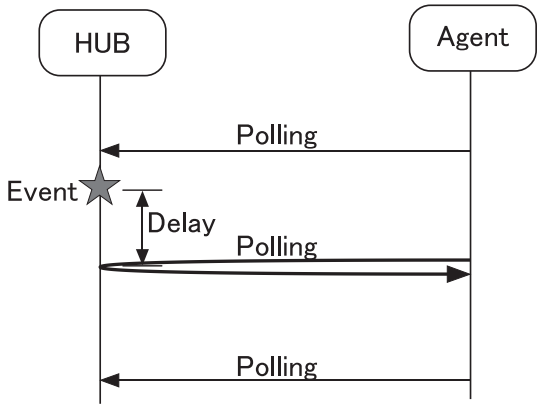

図 2: イベント伝達遅延

を行うまで伝達が遅延する．(図 2)

3.2. 無駄なポーリング

ハブ側で発生したイベントを知るために，ノード側で発生したイベントがなくても頻繁にハブに通信を行う必要がある．このため，意味のない通信が行われる場合がある．(図 3)

ShareTask では，このハブ側イベント取得のための通信をノードの死活監視に利用しているので無駄な通信というわけではない．

しかし，このハブ側イベント取得のための通信のためにトラフィックが増加し，ハブ側 HTTP サーバーのリソースを消費する．さらに，ノードとハブの間に介在するルーターなどのリソースも消費する．

したがって，死活監視とハブ側イベントの取得を如何に効率化するかが課題である．

HTTP は，単方向のプロトコルとして設計されたため，ノードからハブへの通信に限定される．ハブからノードに対して接続することが考えられるが，このような双方向通信を前提にすることは，ハブとノード間のネットワークの接続性に対称性を求めることになり，実際のインターネット上に展開するには困難を伴う．

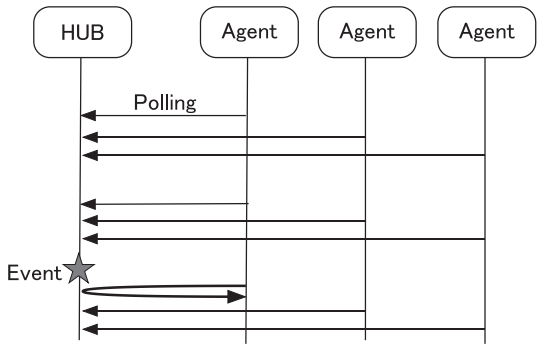

図 3: 無駄なポーリング

4.HTTP 通信のプッシュ化技法

近年，HTTP 本来の通信の単方向性を補い，プッシュ型通信，あるいはさらに進んで双方向型通信を実現する手法あるいはプロトコルが開発されてきた．[5]

4.1.Long Polling

第 1 の手法は，HTTP のプロトコルを修正せずに接続を応答タイムアウト (〜300 秒) まで維持し，その接続時間中に発生したハブ側イベントを応答とすることにより，イベントの伝達遅延を短くする Long Polling と呼ばれる手法である．(図 4) タイムアウトで切断されると，クライアント側（ノード側）は即座に再接続を行うことにより，擬似的に双方向性のある HTTP 接続を実現する．

この手法の欠点は，タイムアウトによる再接続が頻繁に発生するために，HTTP サーバー側（ハブ側）の接続要求受付処理の負荷が高くなることである．

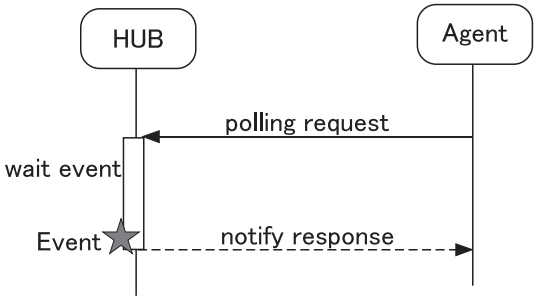

図 4: Long Polling

4.2.HTTP Streaming

第 2 の手法は，HTTP ストリーミングの手法を採用して，HTTP 接続を維持し続けながら，サーバー側でイベントが発生する都度，その通知を応答として送信する方法である．(図 5) この手法によれば，Long Polling で問題となる HTTP 再接続の頻発によるサーバー負荷上昇を避けることができる．ただし，クライアント側からの送信は，HTTP 接続確立時に行えるだけなので，クライアント側で発生するイベントを適宜伝達するためには，別途 HTTP 接続を行う必要がある．

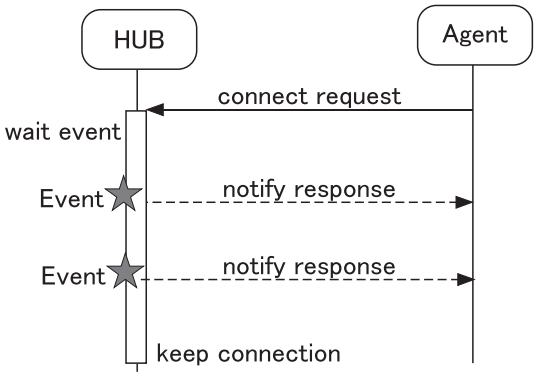

図 5: HTTP Streaming

4.3.WebSocket

第 3 の手法は，WebSocket プロトコル [6] である．WebSocket は，HTTP で確立した接続上を，サーバー側およびクライアント側が双方向かつ非同期でデータフレームを送受信できるしくみである．(図 6) このプロトコルによれば，ひとつの HTTP 接続を維持することで，サーバー，クライアント側双方のイベントの伝達遅延を極めて短くすることができる．

以上述べた 3 つの手法のいずれにおいても，HTTP 接続（TCP 接続）を長時間維持するので，ハブのリソースを消費する．また，ノードが NATP 配下に存在しているばあいには，そのルーターの NATP 変換テーブルのメモリを消費するという問題が発生する．

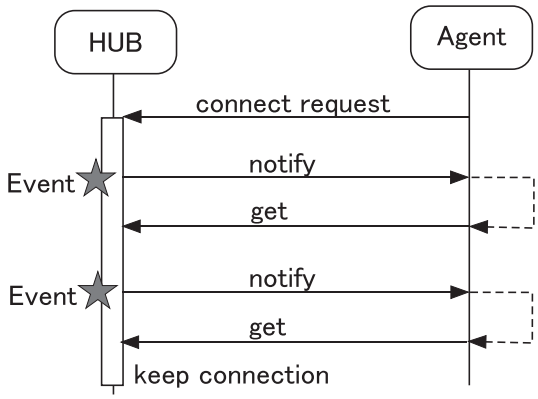

図 6: WebSocket

5. 解決方法

多数のノードからハブへのイベントポーリングのトラフィックを適正なレベルに抑えること，長時間維持される HTTP 接続によるハブ，ルーターのメモリ消費を抑えること，この 2 つ課題について解決方法を考える．

5.1.back-pressure 制御

ノードからハブへのポーリングの時間間隔を動的に制御することにより，ハブに到来するポーリングの頻度を適正な範囲に維持する．

ハブは，ポーリングに対する応答に，ポーリング間隔の指定を加えることにより，各ノードのポーリング間隔を調節し，全体のポーリング頻度を調節することができる．(図 7)

ハブは，ポーリング頻度を常に計測し，自己の処理能力に対して過剰な頻度であれば，ノードに対してポーリング間隔を長くするように指示し，逆に処理能力に余裕がある状況であればポーリング間隔を短くするように指示する．このしくみを back-pressure 制御と呼ぶことにする．

このしくみによって，ハブの応答性能を維持しながら，ハブ側イベントの伝達遅延を可能な限り短くすることができる．

重要なハブ側イベントとして，実行待ちのジョブの存在がある．ジョブを既に実行中で，新たなジョブを実行するための余剰リソースが不足しているノードは，ジョブのポーリングをしないが，そのようなノードが増えてくるとハブに到来するポーリング頻度は小さくなる．この状況では，back-pressure 制御は，ジョブポーリングの頻度を上昇させて，待ち行列上のジョブがノードへの割り当てが行われるまでの待ち時間（待ち行列上の滞留時間）を短くする．一方，待ち行列上にジョブが存在しない状況が続き，ジョブ実行が終了し計算リ

ソースが解放されて新たなジョブを求めるノードが増えてくると，ジョブポーリングの頻度が上昇する．ハブは，ポーリング頻度が自己の処理能力に対して過剰なレベルに近づいてくると，各ノードへの応答においてポーリング間隔を長くするよう指示する．

このポーリング間隔 t の決定は，直近の一定時間区間 (約 100 秒) の中で認識されるノード数 N と，目標とする適切なポーリング頻度 r(回／秒) から単純に $t = \frac{N}{r}$ で求める．r は，あらかじめ決めておく方法の他に，実際にハブの OS 全体の負荷状況から動的に決定する方法も検討中である．

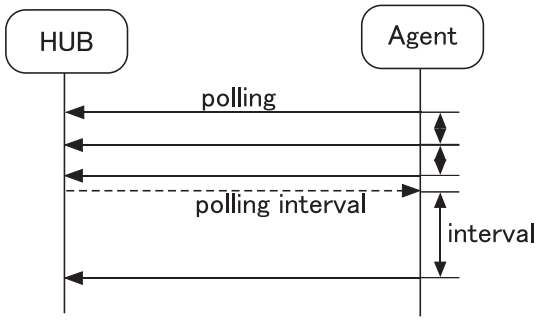

図 7: back-pressure 制御

5.2. シグナリング

WebSocket によって，ハブからノードに対してイベント (実行待ちジョブの発生など) を小さい遅延で伝えることができるが，プル型の特徴はジョブ実行に必要なリソースの有無の判断をノード側が行うことにある．そこで，WebSocket を使用してイベント発生の通知だけを行い，そのイベントに対してのアクションはノード側の判断に委ねる．(図 8) これにより，たとえば，実行待ちジョブの発生イベントを受信したとしても，ノード内計算リソースに余裕がなければ，無視するといった柔軟な制御が可能である．

WebSocket 接続 1 つですべての通信を賄うことも可能だが，ファイル転送など別途 HTTP 接続を行う方が実装上好ましい場合もあり，WebSocket の活用範囲は実装と効果のバランスにおいて検討中である．

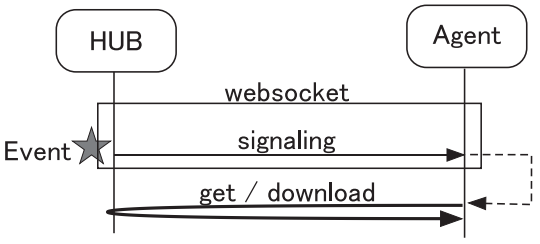

図 8: シグナリング

5.3. 接続の集約

WebSocket を使用した場合，維持される接続数の増大が問題になることがある．ノード群が NATP 配下に配置された場合，ルーター上の NATP 変換テーブルがオーバーフローすることがありうる．そこで，シグナリング用接続を束ねるプロキシーを NATP 配下に配置することによって NATP を越える接続を 1 本に集約する手法を検討中である．(図 9)

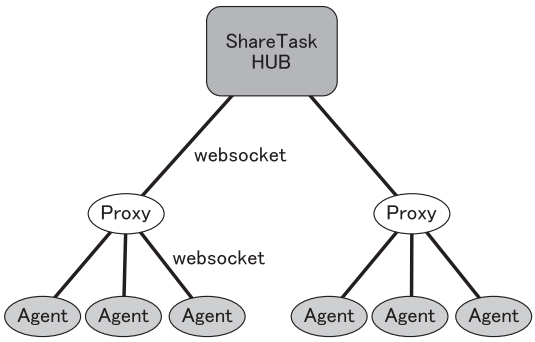

図 9: プロキシー

6. まとめ

HTTP を使用したプル型ジョブスケジューラは，計算ノード数の増大と，分散した拠点の統合に柔軟に対応できる特性を持つが，イベント伝達遅延と HTTP サーバー負荷の 2 つの問題を抱えている．本稿では，HTTP のプッシュ化の技法・プロトコルを整理し，その解決手法を議論した．現在，この検討にもとづいて機能の実現を行っているところであり，パブリックなクラウドサービスなどインターネット上に分散した複数の計算機クラスターの統合運用を実践し，分散した計算資源のジョブスケジューリングについて定量的な評価を報告してゆく予定である．

謝辞

WebSocket の活用について議論していただいた平松和剛，秋本満，千光士生の各氏に感謝いたします．

参考文献

[1] 齊藤隆之, 善甫康成, ”Web ベース計算リソース管理ミドルウェア：ShareTask”, FIT2012 B-012.

[2] 善甫康成, 齊藤隆之, 岡戸晴彦, 近野利信, 千田範夫, ”自律プル型制御方式によるインターネットワイドなジョブスケジューリングの性能試験”, 九州大学情報基盤研究センター 先端的計算科学研究プロジェクト成果報告会 (2010).

[3] 善甫康成, 齊藤隆之, 岡戸晴彦, 近野利信, 古賀良太, 千田範夫, ”メタスケジューリングを指向した計算環境”, 日本コンピュータ化学会 2010 年春季年会研究展示 RX02 (2010).

[4] http://nodejs.jp/ http://nodejs.org/

[5] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins. ”Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP”. Internet Engineering Task Force, Request for Comments: 6202, Apr. 2011.

[6] I. Fette and A. Melnikov, ”The WebSocket Protocol”. Internet Engineering Task Force, Request for Comments: 6455, Dec. 2011.