社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DPS－132 (22)
2007－GN－ 65 (22)
2007－EIP－ 37 (22)
2007／9／21

バッチ制御システムにおける適応的スケジューリング方式の検討

上西 康太 †, 境 美樹 †, 伊織 生美 †

† NTT 情報流通プラットフォーム研究所

業務システムで利用されるバッチジョブのスケジューリングにおいて、バッチジョブの処理時間が毎回変動し、正確な予測が難しい環境の下で、実行すべきバッチジョブの実行実績から実用的なスケジュールを適応的に生成する手法を検討する。提案手法では、同一のジョブは繰り返し実行されるというバッチ処理の性質に着目し、蓄積される実行実績にスケジューラを適応させることで処理時間の予測精度を高めることができる。その上で、確率分布の広がりによって処理時間の変動をモデル化し、スケジューリングの際のマージン設計を容易にすることができると考えられる。

Adaptive Scheduling Method in a Batch Control System

Kota UENISHI † Miki SAKAI † Kiyoshi IORI †

† NTT Information Sharing Platform Lab.

We propose a new scheduling method in a batch control system, which is usually used in business systems. In an environment that processing time of a batch job varies on each execution, sufficient prediction is difficult. Thus we investigate a method to adaptively generate a practical schedule according to the batch job execution history. In our proposed method, we utilyze a characteristic of batch processing that the same job is executed many times to improve the prediction accuracy of the batch execution time. Moreover, we use the spread of probability distribution to efficiently model the variance of batch execution time in order to simplify the design of the margin time of the batch jobs.

1 はじめに

業務系システムにおけるバッチ制御は効率的な実行のために様々なノウハウの蓄積が行われてきた。

これまでは、複数のバッチジョブを一定時間内に終了させるために

1. スケジュールを効率化するために手動で構成または修正する
2. バッチジョブの実行時間を必要以上に長く見積もる
3. 実行時間帯にバッチジョブが終了しない場合のロジックを組み込んでおく

などの対策が施されてきた。

一方で、近年、システムの規模は拡大の一途を辿り、夜間など一定の時間内で大量のジョブの処理が求められる場合、上述の対処だけでは追いつかない場合が発生する。

一例としては、バッチジョブの起動スケジュールの定義は詳細な点まで人手で行われており、それぞれのバッチジョブに対して、アクセスする資源や起動条件などを細かく指定する必要がある。1ジョブ当たりの設定項目数は比較的少ないが、登録するジョブの数が増えるに従って、それぞれのジョブの処理期限を意識したスケジューリングを行うなど、効率的なスケジュール構成のための手順が煩雑になり、手動での管理が困難になる。

そこで本稿では、必要最低限の項目について手動で設定したスケジュール、及びそのスケジュールの実行実績を元に、より効率的なスケジュールを実時間で自動的に再構成することを目的とする。

ここで、スケジュールが効率的であるとは、処

理すべき一定量のバッチジョブが与えられたときに、スケジュール実行中の遊休資源が少ないこと、かつ、全てのバッチジョブ開始から終了までの時間が短いことを指す。

2 業務システムにおけるバッチジョブのスケジューリングの課題

本章では、バッチジョブの静的スケジューリングに対する課題、及び関連研究との差異を明らかにする。

2.1 本稿における前提

より効率的なスケジューリングのためには、実行すべきジョブの処理時間に関する正確な情報が必要であるが、通常はジョブの処理時間は様々な要因によって変動するため、その処理時間に関する正確な情報を入手することは難しい。そこで、処理時間に関する正確な情報が手に入らないということを大前提とし、本稿においては、バッチジョブについて以下を前提として議論を進める。

処理期限 業務システムにおいては、業務の要件からジョブに処理期限が存在する場合が多い。

処理時間の変動 一般にプログラムの実行時間は、データ量、アルゴリズムの計算量、CPU のクロック数やメモリのレイテンシなど様々な要因に依存する。業務系システムにおいて、実行回毎に異なるデータセットを用いること、排他ロックなどの要因からプログラムの実行時間が大きく変動する。

並列処理 多数のバッチジョブに対して複数のバッチ処理資源（CPU/ノード）がある。ここでは簡単のため、バッチ処理資源は全て同じ性能であるとする。

繰り返し実行 バッチジョブは定期的に繰り返し実行される。

依存関係なし 一般的に業務システムにおいてはバッチジョブ間に依存関係が存在することが多いが、本稿では簡単のために依存関係は存在しないこととする。

本稿では、定期的にスケジュールの再構成を行うものとする。例えば、一日単位でバッチジョブが定義されている場合に、一日のバッチ処理の開始時にスケジューラが起動され、前日までの実行実績からジョブモデル適応が行われ、その日のスケジューリングを行うものとする。

2.2 バッチジョブの静的スケジューリングの課題

バッチジョブが開始してから終了までにかかる処理時間は、実行回毎に変動する。その統計的傾向を予測可能な範囲で、有効にスケジューリングに利用するための手法を課題とする。

このような条件下でスケジュールを構成するためには、個々の処理時間の総和としての全体の処理時間の変動に基いて、ジョブを各の処理資源に配置する組み合わせ探索を行う必要がある。

そのために、個々のジョブの処理時間予測から全体の処理時間の変動予測を行う方法や、それを構成する個々のジョブの処理時間の変動の予測を行う方法も課題となる。

これらの課題と、本章で述べた前提に対する解決手法について、3 章で詳しく述べる。

2.3 関連研究

本節では、バッチジョブのスケジューリングが、これまでのスケジューリング研究と異なる前提の下におかれていることを示し、全体を考慮した静的スケジューリングの実現のための課題を整理する。

2.3.1 バッチジョブの処理時間予測

これまでも、リアルタイム OS の分野や、グリッドコンピューティングの分野でジョブの実行時間の予測について多くの研究がなされてきた。リアルタイム OS においては、ジョブの実行時間の確率密度関数を用いてオーバーヘッドを最小にしつつ実行可能なタイムスロットを計算するものなど[1]、ジョブの実行に厳しい時間的制約が課されていると同時に、埋め込み系の分野のハードウェア的制限からジョブの実行データを蓄積しにくいという前提があった。

グリッドコンピューティングの分野では、科学計算における応用を目的としており、ジョブの処理期限が厳しい制約として存在していなかった。

一方、バッチ制御システムにおいては時間的制約が比較的緩やかであり、何度も同じバッチジョブを定期的に繰り返し実行するため、ジョブの実行実績の蓄積が容易である。従って、大量に蓄積されたデータを用いて、バッチジョブの処理時間の予測を改善することが可能となる。

2.3.2 バッチジョブの配置の組み合わせの問題

これまでも、OSの分野や、グリッドコンピューティングの分野ではスケジューリングについて様々な研究がなされてきたが、いずれもジョブ到着時に行われる動的なスケジューリングであり、実行すべきジョブが予め判明していない前提の下での課題であった。

一方本稿の前提によれば、実行すべきジョブが予め全て判明しているため、それらのスケジューリングよりも多くの情報が与えられており、全体を考慮した静的なスケジューリング手法が必要となる。

そのために、まず個々のバッチジョブの処理時間の予測を行い、処理時間予測を全体スケジュールに反映して評価する方法を課題とする。その評価を得ることによって、様々な組み合わせが存在しうるスケジュールを探索することが可能となる。

3 提案手法

本章では、まずバッチジョブの処理時間予測の手法を述べる。バッチジョブの処理時間は変動するものとの前提の下に予測モデルを立てて、同じジョブを繰り返し実行するうちに予測モデルの精度の向上を図る。

次に、このようにして処理時間が予測されたバッチジョブを組み合わせてスケジュールを構成する手法について述べる。

3.1 ジョブの処理時間予測のための確率モデル

バッチジョブ A の処理時間が変動すると仮定し、その確率密度分布を $p_A(t)$ とする。このとき、バッチジョブ A が処理期限時間 T までに終了する確率は、バッチジョブ開始時刻を0とした場合

$$P_A(t<T)=\int_0^T p_A(t)dt \tag{1}$$

と表現される。この $p_A(t)$ の形を求めることがバッチジョブの処理時間を予測することになる。以降、本節ではこれを目的とする。

3.1.1 モデルの選定

本稿の論理は一般の確率過程の分布関数について一般性を失うものではないが、ここでは簡単のため、バッチジョブの終了がポアソン過程によってモデル化できるとし、

$$P_A(t)=\frac{e^{-\lambda_A t}}{k!}(\lambda_A t)^k \tag{2}$$

とする。バッチジョブの終了に関する確率分布なので、通常は $k=1$ とする。ただし、λ_A はポアソン分布のパラメータである。このパラメータを推定することによってジョブの処理時間予測を行う。

3.2 MAP推定による処理時間予測モデルの適応

最新のデータセット $D=\{t_i\}_{i=1}^n$ があった場合に、t_{n+1} を予測する手法のひとつとして、MAP推定がある。MAP推定は、パラメータに事前分布を仮定し、データが与える対数尤度を最大化するパラメータを求める方法である。

$$\hat{\theta}=\arg\max_{\theta}\log P(\theta|D) \tag{3}$$

この値は、ポアソン分布の推定においては

$$\hat{\lambda}=\frac{\alpha+nk-1}{\beta+\sum_{i=1}^n t_i} \tag{4}$$

となる。導出の過程は付録とした。この値によって、ポアソン分布のパラメータが決定され、個々のバッチジョブの処理時間の分布を予測することができる。

これによって、データが蓄積されるに従いポアソン分布のパラメータが変化し、最新のデータに対して適応していくことができる。

3.3 シーケンシャルに実行されるジョブの処理時間の合計の分布

次に、個々のバッチジョブの処理時間の分布を予測した後、複数のバッチジョブをシーケンシャルに実行した場合の処理時間、即ち処理時間の合計の分布を求める過程を述べる。

バッチジョブ A に加えて B が存在する場合、A の直後に B を実行するジョブ $A\to B$ の処理時間の確率密度分布は

$$p_{A\to B}(t)=\int_0^t p_A(t_1)p_B(t-t_1)dt_1 \tag{5}$$

となる。これは、二つのジョブのそれぞれの処理時間の変動が確率分布として与えられた場合に、その処理時間の和の分布を畳み込みによって計算できるためである。

また、関数 f,g の畳み込みを $f*g$ によって表現すると、一般の n 個のジョブのシーケンスに関する処理時間の確率分布は

$$p(t)=(p_1*p_2*\cdots*p_n)(t)$$

と表現できる。この分布を解析的に求めるためには、両辺にラプラス変換 L を施すことによって、畳み込み定理 $L[f*g]=L[f]L[g]$ により

$$L[p(t)]=\prod_{i=1}^{n}L[p_i(t)] \tag{6}$$

として求めることが可能である。

ポアソン分布をモデルとして用いた場合、

$$L[p(t)] = \left(\prod_{j=1}^{n}\lambda_j\right)\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta_{ij}(s) \tag{7}$$

ただし $\lambda_i \neq \lambda_j$ とし、

$$\Delta_{ij}(s) = \begin{cases} \frac{A_j^2}{(s+\lambda_j)^2} & (i=j) \\ \frac{2A_jA_i}{\lambda_j-\lambda_i}\left(\frac{1}{s+\lambda_i}-\frac{1}{s+\lambda_j}\right) & (i\neq j) \end{cases} \tag{8}$$

$$A_j = \prod_{k\neq j}^{n}\frac{1}{\lambda_k-\lambda_j} \tag{9}$$

である。これを逆ラプラス変換することによって、

$$p(t) = \left(\prod_{j=1}^{n}\lambda_j\right)\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta'_{ij}(t) \tag{10}$$

ただし

$$\Delta'_{ij}(t)=\begin{cases} A_j^2te^{-\lambda_jt} & (i=j) \\ \frac{2A_jA_i}{\lambda_j-\lambda_i}\left(e^{-\lambda_it}-e^{-\lambda_jt}\right) & (i\neq j) \end{cases} \tag{11}$$

である。

3.4 バッチジョブの配置の組み合わせの問題

本節では、バッチジョブを各計算資源に静的に配置していくスケジューリングの問題に対するアプローチについて述べる。

3.4.1 問題の定式化

バッチジョブ $J=\{j|j=1,\cdots,m\}$ を複数の処理資源 $W=\{i|i=1,\cdots,n\}$ に、全体の処理時間が短くなるように配置するタスクを考える。ここで、全体の処理時間が最短になる、最適なスケジュールを求める計算は一般に NP-hard 問題であるが、本稿では、ある程度の計算時間で実用的なスケジュールを求めることを目的とする。

ここで、ジョブ j の処理に必要な時間を $T_J(j)$、処理資源 i が引き受けたジョブを全て処理するために必要な時間を $T_W(i)$ とすると、

$$T_W(i)=\sum_j S_{ij}T_J(j) \tag{12}$$

と表すことができる。ここで S_{ij} は、ジョブ j が処理資源 i に配置された場合は1、配置されたなかった場合は0となる値である。従って、最適なスケジュールを求めることは、

$$\max_i\{T_W(i)\} \tag{13}$$

を最小化する $\{S_{ij}\}$ のパターンを求めることである。ただし、ここでは時間の和を用いずに、和の分布を用いるため、実際には $T_J(i)=\log L[p_i(t)]$ として、式(12)で用いる。またこれによって、T_W の意味合いも変わるため、これを評価する際は

$$E[W_i(t)]=\int_0^{\infty}tL^{-1}[e^{T_W(i)}]dt$$

とすればを処理資源 i 全体の処理時間の期待値とし、式(6)の意味を失うことなく計算が可能である。

もしくは別の評価値として、締切時間 T_d に対する処理資源 i の全ジョブの終了確率

$$P_W(T_d)=\int_0^{T_d}L^{-1}[e^{T_W}]dt$$

を用いることも可能である。

3.4.2 組み合わせのアルゴリズム

ここでは、合計の処理時間 $T_W(i)$ が常に最大となる処理資源を検出し、そのボトルネックのジョブを他の処理資源に引き渡すことによって $T_W(i)$ を徐々に短縮していくことが可能になる。

以下に、そのアルゴリズムを示す。

表：ジョブ配置組み合わせの探索

1. 初期配置を適当に決定する
2. $i=\arg\max\{T_W(i)\}$ として、$T_W(i)$ 中の最短時間タスク j と適当に選択した i' に対して、S_{ij} と $S_{i'j}$ の値をスワップする
3. $T_W(i)$ と $T_W(i')$ を求める
4. 評価値 $\max\{T_W(i)\}$ が改善されていなければ、$\{S_{ij}\}$ を2.でスワップする以前の状態に戻す
5. 繰り返し回数が予め設定された制限回数を超えていなければ、繰り返し回数を1増やして2.へ戻る
6. 終了

これは、処理時間が最長となっている処理資源に配置されたジョブを、徐々に他の余裕のある処理資源に配置していく手法である。つまり、処理時間が最長ではない処理資源にはいくらかの時間的余裕があるとして、それに対して可能な限り処理時間の短いジョブを配置していくものである。

これで最適解に到達するという理論的保証はない。従って、繰り返し回数を予め定めておくことによって、ある程度の時間で実用的なスケジュールが得られるようにしておく必要がある。

4 考察

3.1 において、シーケンシャルなジョブについて合計処理時間の変動をポアソン分布によって確率的にモデル化した。これによって、毎回異なるジョブの処理時間の情報を統計的に有効に用いることが可能となる。また、バッチジョブに処理期限があるという前提に対して、この方法によって処理期限に対する客観的な評価尺度を与えることも可能になった。

3.2 では、確率モデルのパラメータを推定する解析的手法を示したが、これは、ジョブを繰り返し実行するというバッチ処理の性質に着目したことによって可能になったものである。これによって、ジョブの処理時間の変動を統計的に予測することが可能になった。しかし一方で、より正確なモデルを必要とする場合には、通常パラメータを増やして対応するが、それによってモデルが複雑化することは避けられず、モデルの正確さと複雑さのトレードオフを考慮した妥協点を決定する必要がある。

3.3 では、複数のジョブの合計処理時間の確率分布を解析的に示した。これによって、変動の幅を確率モデルの枠内で表現することが可能になり、ジョブを配置する際に重要な情報として利用することができる。

3.4 においては、それらのジョブを配置する方法について述べた。並列処理を前提にしていることによって、処理時間を要するジョブが多く割り当てられた処理資源からジョブを振り替え、効率的なスケジュールの計画を容易に立てられるようになった。

5 おわりに

5.1 まとめ

本稿では、バッチ制御におけるスケジューリングについて述べた。

前提として実行すべきバッチジョブが予め判明しており、締切を持った複数のバッチジョブを複数の処理資源に配置するというスケジューリングを行うことを目的とした。そのスケジューリングに対して、バッチジョブの処理時間を確率的に推定しモデルを改善しつつ、バッチジョブの処理時間の合計を確率分布として評価することによって、処理資源に対するバッチジョブの配置の組み合わせを探索するという手法を提案した。

この手法は、同じジョブを繰り返し実行するというバッチ処理の性質に着目し、学習を行うことで、処理時間の予測精度を高めることができ、分布の広がりを考慮することによって、バッチ処理のスケジュールを決定する際のマージンの設計などを容易に行うことができるようになる。

5.2 今後の課題

今後、これらの手法を利用したバッチ制御システムを実装し、ジョブの処理時間予測の精度や、ジョブの配置の組み合わせの計算がどの程度実用的であるか、といったことについて検証する予定である。

実際のバッチ処理においては、特定のジョブの実行が別のジョブの終了に依存する場合や、ジョブの終了状態によってその後のジョブのシーケンスが分岐する場合、また、複数のジョブの終了を待って別のジョブが起動される場合など、様々なケースがある。今後、それらを包括した一般的な場合に関する考察を行う必要がある。

また、ジョブを処理資源に配置していくアルゴリズムにおいては、効率的な組み合わせ探索アルゴリズムが多数提案されており、これらを用いて配置の部分についても効率的に配置を探索できるようにすることができる。

付録 A ポアソン分布の MAP 推定

MAP 推定では、データ D が与える確率モデルのパラメータ θ の尤度 $P(\theta|D)$ を最大化することによって、分布の推定を行う。即ち、ここで、$P(\theta|D)$ はベイズの法則により

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{\int P(D|\theta)P(\theta)d\theta}$$

$$\begin{aligned} &\propto P(D|\theta)P(\theta) \\ &= P(\theta)\prod_i P(t_i|\theta) \end{aligned} \tag{14}$$

となる。これを最大化する $\hat{\theta}$ を求める。

ポアソン分布の場合、通常は事前分布 $P(\theta)$ としてガンマ分布 $g(\lambda|\alpha,\beta)$ を用いる。ここで、α,β はハイパーパラメータであり、この定数の選定も MAP 推定の精度に影響する。式 (14) より、MAP 推定は

$$P(\theta)\prod_i P(t_i|\theta) \propto e^{-\lambda(\sum t_i+\beta)}\lambda^{\alpha+nk-1} \tag{15}$$

を最大にする λ を求める問題となる。ここで、λ を含まない係数は省くことができる。式 (15) を微分することによってによって求められる $\hat{\lambda}$ の最大値は、

$$\hat{\lambda} = \frac{\alpha + kn - 1}{\sum t_i + \beta} \tag{16}$$

となる。

参考文献

1) S. Manolache et al., *"Schedulability Analysis of Applications with Stochastic Task Execution Times,"* ACM Trans. on Embedded Computing Systems, Vol.3, No.4, Nov.2004, pp. 706–735.

2) E. Jeannot et al., *"Improved Rungime and Transfer Time Prediction Meachanisms in a Network Enabled Servers Middleware,"* Parallel Processing Letters, Vol.17, No.1, 2007, pp.47–59.