L-022

大規模 IoT システムにおける計算機リソースサイジングの研究
Computing Resource Sizing Method in Large Scale IoT Systems

小杉 優 † 佐藤 尚也 † 魚住 光成 † 鶴 薫 †
Yu Kosugi Naoya Sato Mitsunari Uozumi Kaoru Tsuru

1. はじめに

システム構築において，ハードウェアの構成，能力といった計算機リソースを決定するためのサイジングはオンプレミスもしくはクラウドのいずれの環境においても，重要な項目である．特に IoT(Internet of Things)向けシステム(図 1)では，扱うデータ量が大きく，調達する計算機リソースも大きくなるため，サイジングの誤りはコストの増大に繋がる．

このとき，単体の性能を元に要件として求められる性能を整数倍にして求め，これに従って計算機リソース の調達を行うことが考えられるが，実際にはリソース競合の影響により，単純な整数倍とならず，調達したリソースが不足，または過剰となることがある．

そのため，適切なサイジングを行うための性能予測は重要である．しかし，多くのシステムは，それぞれの要件に合わせた一度限りの システム構成となるため，過去の実績をそのまま活用することが難しい．

本研究では，アプリケーション構造のモデル式化と，リソース供給の式を待ち行列モデルの組み合わせで行う方法により，計算機リソースサイジング手法の一般化を目的とする．

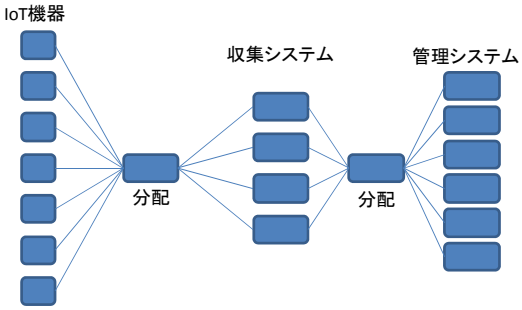

図 1 IOT システムの構成例

2. 先行技術

システムが実現する目的の機能を評価環境で実行してリソースの消費状況を計測し，その消費傾向からリソースの充分性を確認する方法 [1]が取られることがある．この方法によってシステム上のボトルネックを抽出して増強したり，リソースの消費量から評価時よりも高い負荷の実行性を評価する．しかし，システムの性能はアプリケーションの構造的な制約で決定する場合があること，リソース消費とスループットなどの性能は高負荷時は非線形な関係にあることなどから，大規模システムにおいてはサイジングの充分な方法であるとは言い難い．

負荷に応じて性能が非線形に劣化する傾向を計測し，これを多項式で近似することで，異なる環境での性能予測を行う試み[2]も行われている．しかし，多項式に近似するための計測点が多く，また，近似式がカバーできる範囲も限られるという課題がある．

大規模 IoT システムは，多数のデータ発生元が発報するデータを集約する構成を取る事が想定され，集約するサーバへのデータの到着はランダム性があると考えられることから，待ち行列モデルを利用してサーバに求められる処理能力を推定することが適当である．

待ち行列に関わる研究は電話交換を対象とした Erlang の研究[3]が端緒であるが，1950 年代以降，D.G.Kendall の論文[4]を契機に本格化したといわれている．情報処理の世界では，1980 年代に Word Whitt の Queueing Network [5]や，Stephen S. Lavenberg の性能モデル[6] など，複雑化したシステムの性能についての研究がすすめられた．

待ち行列モデルは古典的な性能モデルであるが，今日の情報システムのシステム設計においては，構造の複雑さによる適用の煩わしさと，ブラックボックス化したモジュールの増加の為，利用が進んでいるとは言い難い．

3. 課題

到着したデータに対して多段の処理を実行するアプリケーション構成を例として取り上げる (図 2).

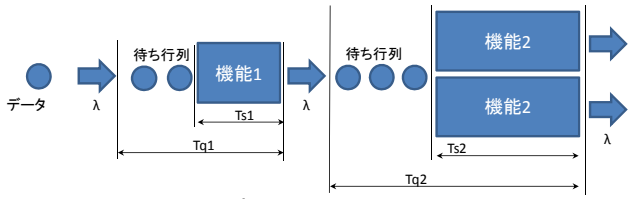

図 2 アプリケーション構成例

このとき，データ到着から出力までの平均時間 Tq (= Tq1 + Tq2)は，待ち行列モデルにより算出できる．処理要求はランダムに到着するとして待ち行列モデルに基づくモデル式を，

平均時間 = mmn(窓口数，到着率，平均処理時間)

と表すと，

Tq = mmn(1, λ , Ts1) +mmn(2, λ , Ts2)

となる．これにより，システムの要件であるλに対して，妥当な Tq が得られれば良いが，機能 1 と機能 2 は同じサーバ上で動作し，また，同じストレージを利用する場合，

† 三菱電機株式会社， Mitsubishi Electric Corporation

サーバのコア数やストレージの性能が評価に反映されないという課題がある．

アプリケーションの構造を詳細に分析し，具体的なリソースアクセスまで明らかにすることで，精度の高い性能のモデル式を作成することは原理的には可能であるが，アプリケーションの複雑化とブラックボックス化により困難である．

本研究では，概念的なアプリケーション構造とリソースの原理的な動作モデルから，システムのサイジングが可能なモデルによって課題の解決を図る．

4. 解決策

4.1 サーバ性能特性のモデル式化

アプリケーションの構造に沿った待ち行列モデルと，アプリケーションのリソース要求の待ち行列モデルによる性能特性を複合することで，サーバの性能特性をモデル化する(図 3).

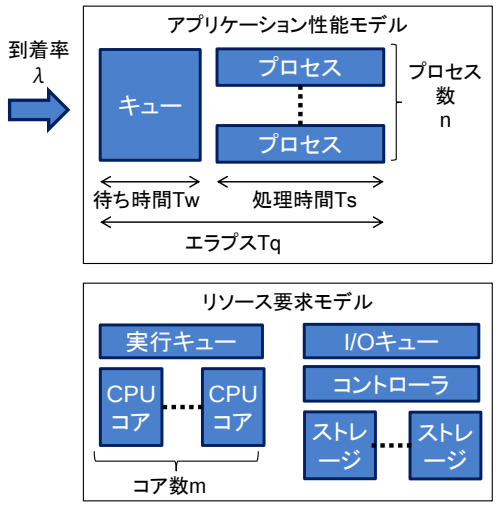

図 3　性能特性モデル

4.1.1 アプリケーション性能モデル

平均処理時間 Ts 秒リソースを消費してプロセスを実行するアプリケーションを考える．

アプリケーションは処理要求に対して 1 つのキューを持ち，1 つのキューが n 個の独立したプロセスに処理を分配する構造であるとする(図 4）．

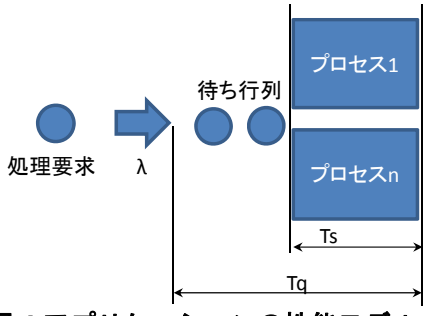

図 4 アプリケーションの性能モデル 1

このとき，到着率λ件/秒の処理要求に対するエラプス(待ち時間+平均処理時間)Tq 秒は，待ち行列モデルを用いて式(1)で表わせる．

$$Tq = mmn(n, \lambda, Ts) \quad \cdots \text{式(1)}$$

アプリケーションはプロセス毎にキューを持ち，到着した処理要求はそれぞれのキューに均等に振り分けられる構造であるとする (図 5）．

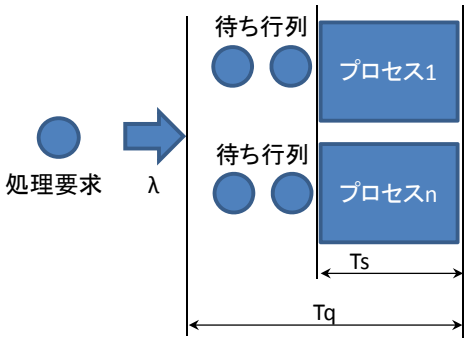

図 5 アプリケーションの性能モデル 2

このとき，到着率λ件/秒の処理要求に対するエラプス(待ち時間+平均処理時間)Tq 秒は，待ち行列モデルを用いて式(2)で表わせる．

$$Tq = mmn(1, \lambda/n, Ts) \quad \cdots \text{式(2)}$$

アプリケーションが p 個の処理単位から構成され，処理間にキューが配置されている場合，その性能特性は mmn() の組み合わせで表現できる．

4.1.2 リソース要求モデル

式(1)，式(2)における平均処理時間 Ts は，複数のリソースの消費，また，リソース の混雑具合によって変動する．

使用するリソースが CPU とストレージであるとし，それぞれの混雑具合の結果の待ち時間を含む時間であるとすると，Ts は式(3)のように表す事ができる．

$$Ts = Tq\_cpu + Tq\_io \quad \cdots \text{式(3)}$$

アプリケーションのプロセスは連続的にリソースを使用するわけではないが，図 6 のようにモデルを単純化する．

単純化することで，n 個のプロセスが Ts_cpu 分の CPU と Ts_io 分のストレージアクセスを消費すると考える事ができる．このときの Tq_cpu と Tq_io は，図 7，図 8 のような原理的な動作モデルとなる．

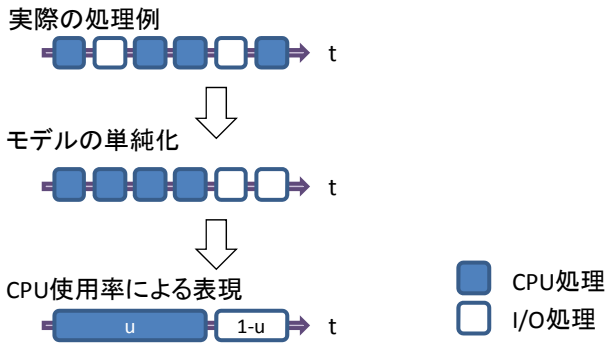

図 6 リソース消費の単純化したモデル

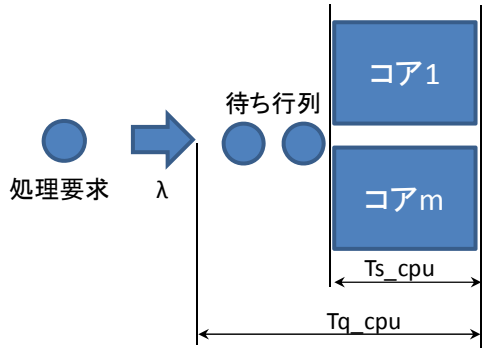

図 7 CPU の原理的な動作モデルと処理時間

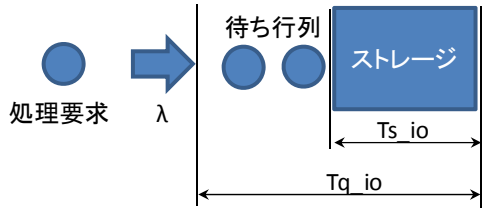

図 8 ストレージの原理的な動作モデルと処理時間

性能のモデル式は，それぞれ式(4)，式(5)のように表す事ができる．

$$Tq\_cpu = mmn(m, \lambda, Ts\_cpu) \quad \cdots 式(4)$$

$$Tq\_io = mmn(1, \lambda, Ts\_io) \quad \cdots 式(5)$$

OS の管理する CPU の Run Queue はコア毎におかれる実装が多く，図 7 とは異なるが，Run Queue 間の調停が行われるためコア数を窓口数とした図 7 のモデルに沿った式(4)の性能特性となる．

ストレージについては，多段のキャッシュを持っているため，Ts_io は安定しないが，最大負荷時はハードウエアデバイスのヘッドシークや回転待ち時間に拘束される．負荷が上昇するにつれこの値に収束する．

4.2 性能モデル式の決定

アプリケーションの構造をモデル化して多重実行時の性能特性を明らかにするとともに，そこで発生するリソース要求についてリソース毎のモデル化を行い，両者を組み合わせることで，実行時間の予測を可能とする方法を採る．

アプリケーション構造のモデル式化は，処理の実行主体であるプロセスの要求を受け付ける機能の構造，構成するプロセスの相互の接続の構造に着目し，キューの有無，個数，キューのつらなりの数により式(1)，式(2)の組み合わせを考える．

例えば，図 2 のようなアプリケーションの構造の場合，アプリケーションの性能モデルは，式(1)，式(2)を組合せて式(6)のように表す事ができる．

$$Tq = mmn(1, \lambda, Ts1) + mmn(2, \lambda, Ts2) \quad \cdots 式(6)$$

ストレージのアクセスが機能 2 で Ts_io 行われるとし，CPU のコア数は m であるとすると，リソース要求の性能式は，

$$Tq1\_cpu = mmn(m, 2\lambda, Ts\_cpu) \times Ts1\_cpu/Ts\_cpu$$

$$Tq2\_cpu = mmn(m, 2\lambda, Ts\_cpu) \times Ts2\_cpu/Ts\_cpu$$

$$Tq2\_io = mmn(1, \lambda, Ts\_io)$$

となる．従って，

$$Ts1 = Tq1\_cpu$$

$$Ts2 = Tq2\_cpu + Tq2\_io$$

$$Ts\_cpu = Ts1\_cpu + Ts2\_cpu$$

より，全体の性能式は，

$$Tq = mmn(1, \lambda, mmn(m, 2\lambda, Ts\_cpu) \times Ts1\_cpu/Ts\_cpu) + mmn(2, \lambda, mmn(m, 2\lambda, Ts\_cpu) \times Ts2\_cpu/Ts\_cpu + mmn(1, \lambda, Ts\_io)) \quad \cdots 式(7)$$

と表す事ができる．

この性能式によって，Ts1_cpu，Ts2_cpu，Ts_io を仮定することで，λに対する Tq を算出することができる．

5. 評価

前章で検証した性能モデルについての妥当性 を検証した．サンプルアプリケーションを元に，モデルによる理論値と，実機での実測値との比較を行った．

5.1 サンプルアプリケーションの構成と性能式

サンプルアプリケーションは図 9 のように複数の機能が連携し，且つ，それら機能間で後段の完了を待つ処理が 1 つのサーバ上で複数動作する構成である．リソースはマルチコアの 1 台のサーバであり，ストレージは，システムで 1 つの構成である．

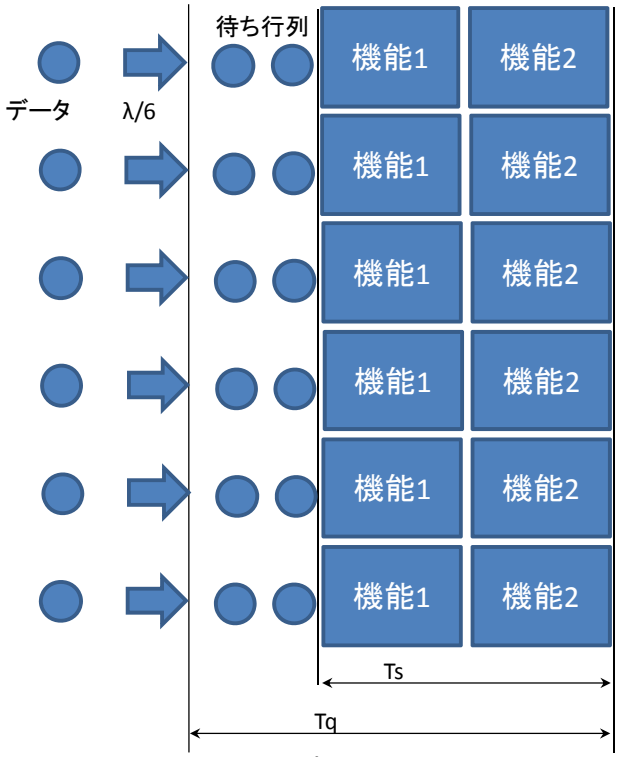

図 9 評価対象のアプリケーション構成

このアプリケーション構造の性能式は，式(8)となる．

$$Tq = mmn(1, \lambda/6, Ts) \quad \text{・・・式(8)}$$

リソース要求の性能式は，

$$Ts = Tq\_cpu + Tq\_io$$

$$Tq\_cpu = mmn(m, \lambda, Ts\_cpu)$$

$$Tq\_io = mmn(1, \lambda, Ts\_io)$$

であり，全体の性能式は，

$$Tq = mmn(1, \lambda/6, mmn(m, \lambda, Ts\_cpu) + mmn(1, \lambda, Ts\_io)) \quad \text{・・・式(9)}$$

となる。

5.2 計測結果の評価

対象のアプリケーションでの測定は，コア数 m を m =2,3,4,6 と変えて計測をおこなった．それぞれの到着率λとエラプス Tq をプロットした．図 10 中の点がそれにあたる．

また，式(9)に一定の Ts_cpu，Ts_io を与えて，m =2,3,4,6 としたときの演算結果を図に描画している．図 10 の曲線がそれにあたる．

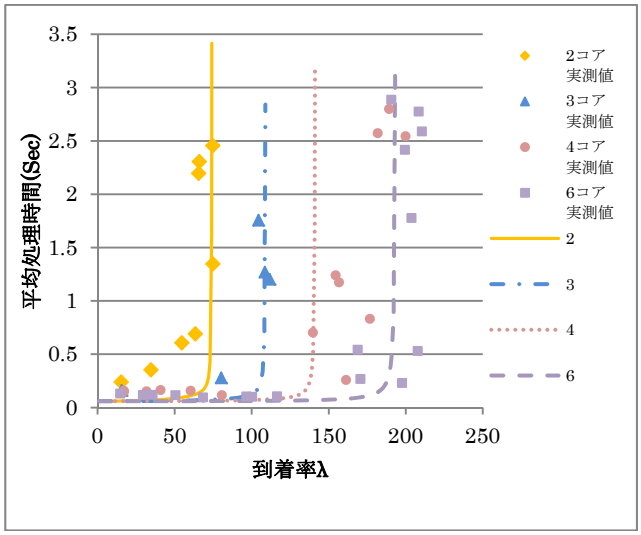

図 10 到着率Λとコア数の関係

式(8)によって描かれた曲線は，それぞれのコア数において，実測値が示す性能劣化，限界性能に沿っており，おおよそ実測値の傾向をとらえているといえる．

この方法を利用することで，サイジングにおいては，小規模な構成における計測から適当な Ts_cpu，Ts_io を決定し，コア数の変更や Ts_io を短縮することによるより早いストレージを採用した場合の試算などを行うことで，より適切なリソースの規模，能力の決定を行う事ができるといえる．

6. おわりに

大規模システムにおけるサイジングは，オンプレミス環境、クラウド環境いずれに対するシステム構築であっても大きな課題である．

本研究では，アプリケーション構造の性能とリソースの性能を，待ち行列モデルを使って数式化し，両者を組み合わせることでシステムの全体の性能を試算する方法を提示した．

サンプルアプリケーションを使って性能式を評価し結果，概ね実測値の傾向がつかめていることが確認された．この方法は，より適切なリソースの規模，能力の決定を行う事ができるといえる．

今後は，様々な形態のアプリケーションへの適用と評価を進め，汎用性のある手法となり得るか検証を進めていく．

参考文献

[1] Ian Molyneaux，アート・オブ・アプリケーション パフォーマンステスト，オライリージャパン，2009
[2] 市原利浩ら，予備系システムのダウンサイジング手法及び評価，FIT2014（第 13 回情報科学技術フォーラム），2014
[3] Erlang,A.K.: Probability and Telephone Calls, Nyt. Tidsskr. Mat. Ser.B,vol.20, pp.33 － 39，1909
[4] Kendall,D.G.: Some Problems in the Theory of Queues, J.Roy.Statist.Soc.,Ser.B, vol.13,no.2,pp.151.158,1951
[5] Word Whitt : The Queueing Network Analyzer, THE BELL SYSTEM TECHNICAL JOURNAL, Vol.62, No.9., 1983
[6] Stephen S. Lavenberg, COMPUTER PERFORMANCE MODELING HANDBOOK, ACADEMIC PRESS, 1983