RO-001

リソース不足に起因する Web システム障害のストリーム DB を用いた予兆検知
Prediction of Web Application Failure Caused by Resource Shortage Using Stream DB

吉野 松樹(1)　薦田 憲久(2)　半田 敦郎(1)　大場 みち子(3)
Matsuki Yoshino　Norihisa Komoda　Atsuro Handa　Michiko Oba

1. 背景

インタネット上で公開される Web システムは，アクセスの増加等の理由でサーバのリソース不足が発生しシステム障害となる場合がある．運用時にリソース不足が発生しないように，システム設計時点でシステムの性能要件を満たすために必要な稼働環境の規模を見積もるキャパシティプラニングが一般的に行われている．キャパシティプラニングの例として，世界最大規模の写真投稿サイトにおいて，増加し続けるトラフィックに対応するために追加が必要なリソースを見積るための手法の報告がある(1)．Web システムの性能要件には，応答性能，単位時間あたりの処理件数，処理すべきデータ量などがある．キャパシティプラニングの方法として，類似のアーキテクチャで構築された Web システムの実績を参考にすることが一般的に行われている(2)．例えば，Web アプリケーションに対して，情報参照系，情報更新系といったアーキテクチャ毎に，リクエスト毎のメモリ，CPU といったリソースの使用量を実績をベースに基礎値として定め，これと性能要件から必要となるサーバ等の稼働環境を見積もるといったことが行われている．この見積もりに対し，システムを構成するアプリケーションがほぼ完成した時点でシステムテストの一環として負荷テストを実施し，見積りの妥当性の検証を行い，用意されている稼働環境の規模が適当であるかどうかを確認する．

このようにしてキャパシティプラニングを実施しても，運用中にリソース不足が発生し，システム障害となる場合がある．リソース不足の発生の予兆を検知することで，障害発生に至らないようリソースを追加したり，新規のアクセスリクエストを制限するなどの運用を行うなどの回避策を採ることが可能となる．近年，リソース不足が発生すると自動的にシステム構成をスケールアウトする機能を有するクラウドサービス(3)が提供されているが，スケールアウトによって利用するサーバ数が増加すれば利用料金も増加するので，クラウドサービスの利用料金を管理するためには，リソース不足の発生の予兆検知は有用である．

本論文では，リアルタイムに時系列データの解析を行うことができるストリーム DB を利用してシステムの状態を監視し，リソース不足となる予兆を検知する方法を提案する．

2．キャパシティプラニングの方法と問題点

2.1　キャパシティプラニングの方法

Web システムを対象としたキャパシティプラニングの例について説明する．

(1)（株）日立製作所ソフトウェア事業部
(2) 大阪大学大学院情報科学研究科
(3) 公立はこだて未来大学情報アーキテクチャ学科

表１．キャパシティプラニングで用いるパラメータ例

	種別	項目	単位
1	システム要件	最大同時ログイン数	人
2		1 ユーザの利用時間	分
3		1 ユーザのリクエスト回数	回
4		1 秒あたりの最大リクエスト数	件/秒
5		目標レスポンス時間	秒
6		内部保留時間	秒
7		参照業務割合	%
8		更新業務割合	%
9	業務の特徴	最大表示項目数	件
10		最大表示行数	件
11		最大表示データ数	文字
12		最大ユーザ入力情報量	文字
13		最大ユーザ入力情報数	件
14	マシン仕様	SPECInt_rate_base2000	

大部分の Web3 階層アプリケーションは，情報参照系処理，情報更新系処理の組合わせで実現されている．適切なアプリケーションフレームワークを利用してアプリケーションを開発した場合，システムで必要なリソース量は表１に示すようなシステムの外部的な要件と特徴を表すパラメータと，アプリケーションフレームワークの内部構造から決まる参照・更新などの処理の単位毎に必要となるリソース所要量から算出することができる．算出されたリソース量に一定の安全係数を掛け，利用可能なサーバの仕様を勘案して用意するサーバ台数を決定する．

例えば，単純化のために性能要件として１秒あたりのリクエスト数 n を考える．使用するアプリケーションフレームワークによって決まる処理単位毎のリソース所要量(例えばメモリ使用量など)として 1 リクエストあたりのリソース所要量を r とし，リクエスト数に依存せず固定的に必要とするリソース量を F とするとき，システムとして必要なリソース総量 R は，

$$R = F + nr \quad \cdots\cdots (1)$$

で算出できる．この場合，想定される 1 秒あたりの最大リクエスト数を N とするとき安全係数を $\alpha\,(>1)$ とすると，準備すべきリソース総量 $R_{\max}$ は，次の式で与えられる．

$$R_{\max} = \alpha(F + Nr) \quad \cdots\cdots (2)$$

2.2　キャパシティプラニングの問題点

前節で述べた方法によるキャパシティプラニングを行った場合に以下のような状況が発生するとリソース不足に起因するシステム障害が発生する．例として，性能要件として同時アクセスリクエスト数を考える．

(1)　同時アクセスリクエスト数が想定していた最大数 N を超えてしまう場合.
(2)　式(1)の r が過小評価の場合.
(3)　必要リソース量が式(1)のような n の 1 次式で与えられる値を超えてしまう場合.

上記の(1)の場合には，同時アクセスリクエスト数あるいは直接リソース消費量の計測を行い，それに基く予測を行うことでリソース不足発生の予兆検知を行うことができる.
上記の(2)(3)は，アプリケーションが完成した時点で実際に稼働させ，複数の同時アクセスリクエスト数に対してリソース消費量を計測することで，r の値が当初の想定値どおりになっているかどうか，あるいは式(1)が成立しているかどうかを確認することで，ある程度防止することができる．しかし，テストにおいて r の値が妥当であり，式(1)が成立していることが確認できていてもアプリケーションプログラムの不具合その他実行環境に起因する問題により，上記(2)または(3)の状況が発生する場合がありうる.
現在多くのシステムでは，リクエスト，リソース使用量など各種のログ情報を取得し，定期的に解析することで，キャパシティプラニングにおける見積りの妥当性の検証を行い，リソース追加計画の検討を行ったり，障害が発生した場合にログの情報を解析し，どこに問題があったかを分析することが行われている．ログ解析によって判明した障害発生前の状況の発生を，リアルタイムにログの情報を解析し検知することで，同様の障害発生を事前に検知することができる．また，類似のシステムでの過去の障害発生事例から一般的なログ解析パターンを抽出し，リアルタイム監視することにより，障害の発生を事前に検知することもできる．次章以降で，ログ情報をリアルタイムにストリーム DB で解析することによって稼働中のシステムに対して，リアルタイムにリソース不足に起因する障害発生の予兆検知する方法を説明する.

3. ストリーム DB の適用によるリアルタイム障害予兆検知

3.1　ストリーム DB の概要

ストリーム DB は，刻々と生成されるデータ系列の分析において，近年注目されている技術である．ストリームは，発生順に追加される無限に続く時系列データ系列である．ストリームから一定の条件に基づいて有限個数のデータ系列を抽出した結果は RDB におけるテーブルとみなすことができる．ストリーム DB では，この有限個数のデータ系列に対して，RDB における SQL（Structured Query Language）と類似の構文とセマンティクスを持つ問合わせ言語 CQL（Continuous Query Language）[4]によって問合わせを行うことができる.
ストリーム DB の適用事例として，ネットワーク監視に適用した AT&T ベル研の Gigascope[5]などの事例，データセンタにおけるメッセージ監視に適用した例が報告されている[6]．また，ストリーム DB をネットワーク監視や，ネットオークションの状況監視に適用する場合の CQL の例が Stanford University stream query repository[7]に示されている.

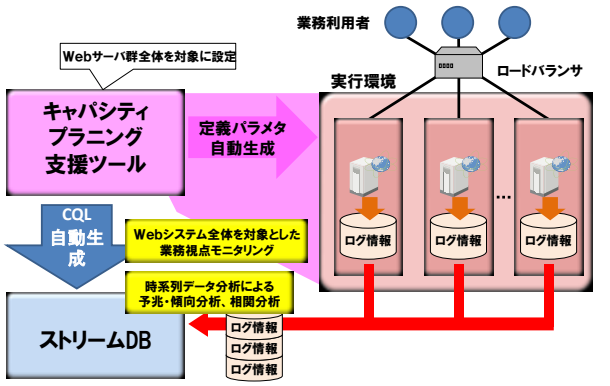

図 1. ストリーム DB によるリアルタイム予兆検知方式概要

3.2　ストリーム DB を利用したリアルタイム障害予兆検知システムの概要

図 1 に本論文で提案するストリーム DB を利用したリアルタイム障害予兆検知システムの概要を示す．1 章で述べた方式に基いたキャパシティプラニングが，入力されたパラメータから計算式に基いて必要なサーバ台数等を算出するキャパシティプラニング支援ツールを用いて行われることを前提としている．キャパシティプラニング支援ツールは，実行に必要なリソース量を確保するために必要な実行環境の定義パラメータの生成も行う.
キャパシティプラニングの結果に基き，ログ情報として得られるデータの値を分析し障害の予兆が検知された場合にはアラームを上げるための CQL をキャパシティプラニング支援ツールで自動生成する.
実行環境では，各種のログ情報を取得するが，このログ情報をファイルに蓄積すると同時にストリームとしてストリーム DB の入力とし，自動生成された CQL によって障害の予兆検知を行う.
以降の節で，予兆検知手法，CQL 自動生成，ログデータの取込み，予兆検知の精度向上のための改善点について説明する.

3.3　予兆検知手法

従来よく用いられているのは，リソース消費総量の閾値監視によるリソース不足の予兆検知である．しかしこれでは，急激なリソース消費量の増加には対応が難しい．この課題を解決するための，ストリーム DB によるログ情報の時系列分析による予兆検知を以下に説明する.
まず，注目しているリソースの消費総量の増加率に着目する方法が考えられる．すなわちリソース消費総量の移動平均の増加率と現在のリソース消費総量から閾値越えを線形外挿によって予測する方法である．CQL を用いて，移動平均及びその増加率の計算及び線形外挿により，直近の傾向が続いた場合に予め定められた閾値を越えることを検知することは容易である．しかし，この方法ではキャパシティプラニングでの想定と何が違ってリソース消費量が当初の見積りを越えたのかがわからないという欠点がある．適切な対応を取るためには，キャパシティプラニングの想定通りにシステムが稼働しているかどうかを監視し，想定と異なっている場合にその部分を明確にすることが必要である.

2 章で示したキャパシティプラニングの想定通りとならない場合の(2)あるいは(3)の場合には，キャパシティプラニングの前提条件が成立していないということであり，早期にその状況を検知し対策を打つことが必要である．2 章で示した例で考えると，式(1)を変形して次の式が得られる．

$$r = \frac{(R - F)}{n} \quad \cdots\cdots(3)$$

システム稼働中のリソース消費総量 R_m（例えば，メモリ消費量，CPU 消費量など）と同時アクセスリクエスト数 n をログ情報から取得することで，1 リクエストあたりの実際のリソース消費量 r_m は(4)式となり，リアルタイムに CQL で計算できる値である．

$$r_m = \frac{(R_m - F)}{n} \quad \cdots\cdots(4)$$

右辺を計算して得られる値 r_m を実測基準値と呼ぶ．キャパシティプラニングの想定通りにアプリケーションが動作していれば，$r = r_m$ が成立し，設計基準値と実測基準値が一致する．安全係数 α を考慮に入れると，次の不等式が成立している必要がある．

$$r \geq \frac{\left(\frac{R_m}{\alpha} - F\right)}{n} \quad \cdots\cdots(5)$$

R_m, n はログ情報からリアルタイムに取得できるので，式(5)が成立しているかどうかを監視することで，システムがリソース消費に関してキャパシティプラニングでの想定通りの振る舞いをしているかどうかを判断することができる．式(5)が成立している状況においては，最大同時アクセスリクエスト数が当初の想定範囲内であればリソース不足は発生しないので，同時アクセスリクエスト数，あるいはリソース消費総量の最新のデータとそれに基く線形外挿によってリソース不足を予測することができる．

逆に式(5)が成立しない場合には，キャパシティプラニングの前提条件が成立していないことを意味するので，リソース消費総量が閾値を越えていなくても，アプリケーションの見直し，性能要件の見直し等が必要となる．同時アクセスリクエスト数の制限や，処理待ちリクエストのキューの長さなどをパラメータの設定で制御しているシステムでは，これらのパラメータの設定を見直すことで，リソース不足に起因する障害を未然に防ぐことが可能である．

さらに，式(5)が成立してるかどうかを判定するだけでなく，式(4)で与えられる r_m の変動を時系列的に評価することにより，リソース消費に関してシステムが安定的に想定通りの振る舞いをしているかどうかを判定することができる．式(4)で与られる r_m の移動平均と分散を計算しその時系列での傾向を評価することによってシステム動作の安定性を評価することができる．移動平均と分散が一定であれば動作は安定していると評価できる．移動平均が増加傾向にある場合には，式(5)が成立しなくなる状況を予測することができる．また，分散が閾値を越えている場合には，システムのリソース消費の予測が難しいことを示しており，アプリケーションおよび実行環境の振る舞いがキャパシティプラニングの想定と異なっていることを示している．このように，リアルタイムに監視することで，実際にシステム障害が発生する前の予兆となる状況を事前に検知することが可能となる．

本節で説明した予兆検知手法をまとめると，以下のようになる．

(1) 着目するリソース消費総量の閾値越えの線形外挿による予測．
(2) 実測基準値の設計基準値からの偏差の監視．偏差が想定範囲内ならば，(1)の線形外挿予測が有効である．
(3) 実測基準値と設計基準値の偏差が想定を越えることの線形外挿による予測．実測基準値の分散が一定の値以内に収まっていることの監視によるキャパシティプラニングそのものの妥当性の評価．

さらに，Web システムにおいては，複数の Web サーバ，アプリケーションサーバが使用されるのが一般的であるので，上記(1)(2)(3)について各サーバ間でのばらつきが一定の範囲内に収まっているかどうかを監視することで，それぞれのサーバだけでなくシステム全体として想定通りに稼働しているかどうかを監視する．

3.4 CQL 自動生成

3.3 で説明した予兆検知手法を実装する CQL は，キャパシティプラニングの情報を基にして自動生成する方法について述べる．

3.3 の(1)の線形外挿による予測の場合は，監視対象となるリソース消費量，監視時間（あるいは監視データ数），およびリソース消費量の閾値の情報があれば CQL を生成することができる．監視時間あるいは監視データ数は CQL が処理対象とするスライディングウィンドウの定義を定める．定義されたスライディングウィンドウ内でのデータの増加率の算出及び最新値と増加率から閾値越えを判定する CQL は，監視するデータによらず同じ構造である．CQL の生成のためには，入力ストリームの特定が必要である．これに関しては，3.5 節で説明する．

3.3 の(2)の基準値相当値の監視についても，式(4)の右辺相当の計算式はキャパシティプラニング時の計算式から導かれるため，ログ情報を入力するための入力ストリームの特定が行えれば，キャパシティプラニング支援ツールが保持している情報から生成することができる．

3.3 の(3)についても平均，分散の計算を CQL の組み込み関数で行うことで(2)の場合同様入力ストリームの特定が行えれば CQL を生成することができる．

複数のサーバ間でのばらつきを評価する CQL についてもそれぞれのサーバで計算した値を再びストリームとして出力し，それらをマージしたストリームに対して，平均と平均からの偏差を計算する CQL として生成することができる．対象となるサーバの個数はキャパシティプラニングの結果として求められるものであり，入力となるストリームは，キャパシティプラニング支援ツールが生成したクエリの結果となるため，その名称もわかっている．

キャパシティプラニング支援ツールからリソース不足予兆検知用の CQL を自動生成することで，情報の一元管理が図れ，CQL 作成・テストの工数を削減できる．

3.5 ログ情報の取り込み

ログ情報はストリーム DB を意識することなく，通常通りファイルに書き込まれるが，ファイルへの書き込みを一定時間毎に監視しストリームに変換するファイルアダプタ

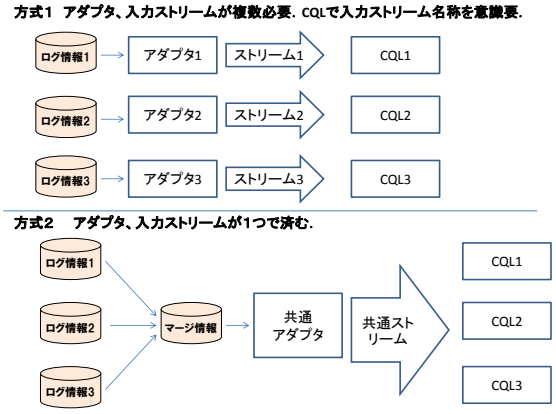

図 2. ログデータ取り込み方式の比較

を利用してストリーム DB への入力とする．

各種存在するログファイルの種別毎にファイルアダプタを作成し，それぞれ個別の入力ストリームとするやり方も考えられるが，この方式では必要となるファイルアダプタの数と入力ストリームの数が増えるという欠点がある．3.3 で説明したように CQL の自動生成において，キャパシティプラニング支援ツールだけでは決定できない情報は，各種のログデータを取り込むためのストリームの名称であり，この方法ではどのログデータを対象とするかによりストリームの名称が異なり，管理が煩雑となる．(図 2 の方式 1 参照．)

CQL では，入力ストリーム中に CQL で参照されないデータがあっても特に問題はない．そのため，複数のログデータの情報をマージしたファイルを一度生成し，そのファイルに対してファイルアダプタを作成し，ファイルアダプタの数と入力ストリームの数を減らすという方式を採用する．(図 2 の方式 2 参照．) ログデータのマージのために若干のタイムラグが発生するが，ファイル書き込みの監視時間を十分短く設定することにより，予兆検知には実用上問題のないレベルとなる．

上記の方式をとることにより，入力ストリームの名称を「固定的な名称＋サーバを識別する名称」という形式にできる．キャパシティプラニング支援ツールでは，入力ストリーム名称として共通的にこの名称を使って CQL の生成を行えばよい．CQL 中で参照するデータの名称は，ストリーム定義中で定義されたデータ名称を用いる．

4. プロトタイプ実装による実験

本論文で提案した予兆検知システムのプロトタイプを構

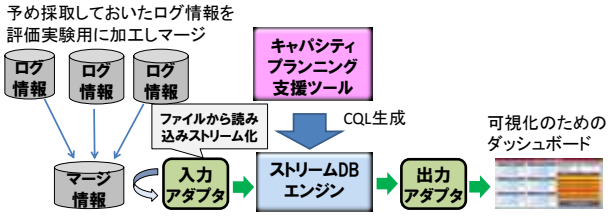

図 3 .プロトタイプ実験環境の概要

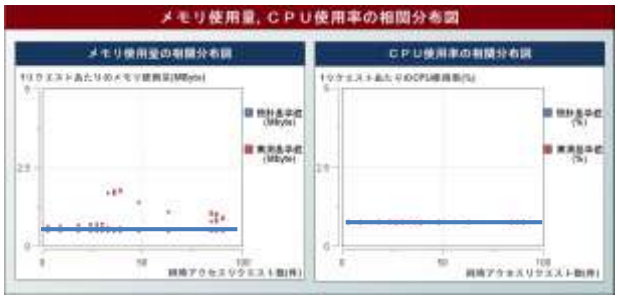

図 4. 設計基準値と実測基準値の乖離による障害予兆検知例

築し，予兆検知の実験を行った．図 3 にプロトタイプ実験環境の概要を示す．解析対象のログ情報は，利用者数，同時アクセスリクエスト数，メモリ使用量，CPU 利用率の 4 種類である．予め実行して取得済みの 4 種類のログ情報を 3.5 で説明した方式を適用できるようにマージし，実験用に加工して，Web/AP サーバ単位のログ情報を作成した．さらに，3 台の Web/AP サーバ環境を想定して 3 つのログ情報をマージした情報を入力アダプタの入力とした．ログ情報をマージすることで，4×3=12 個のストリームを扱う代わりに 1 つのストリームだけを扱えばよく，入力アダプタも共通アダプタ 1 つで処理が可能となっている．入力アダプタにより，マージされたログ情報をファイルから読込みストーリム化し，キャパシティプラニング支援ツールの定義情報から自動生成した CQL を用いて解析を行い，出力アダプタ経由で解析結果をダッシュボードで可視化した．

1 リクエストあたりのメモリ使用量，CPU 使用率を基準値として，設計基準値と実測基準値の乖離による障害予兆検知を行っている状況のダッシュボードでの表示を図 4 に示す．

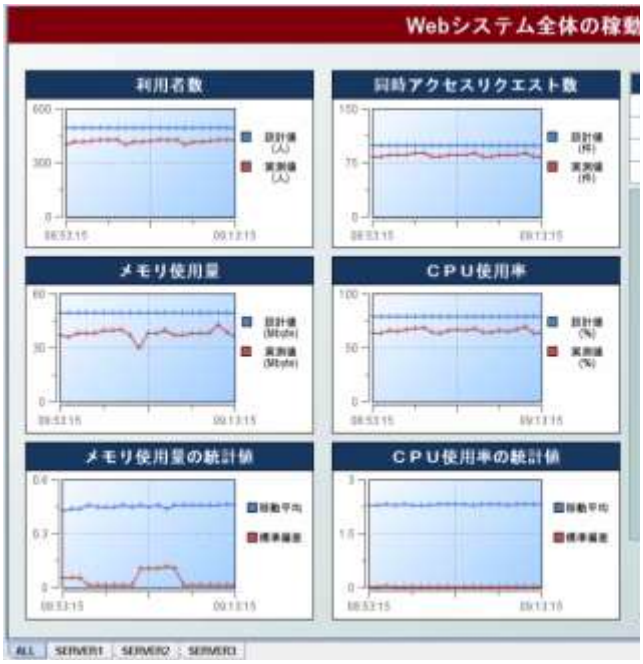

図 5. 実測基準値が設計基準値以内となっている例

図 4 の上の画面で左側は，リソース量としてメモリ使用量，右側は CPU 使用率を示している．いずれも実線で設計基準値を示している．この例では，メモリ使用量の実測基準値が設計基準値を大きく上回っているところがあり，この段階でメモリ不足に起因するシステム障害の発生の予兆を検知することができる．図 4 の下の画面の左側を見ると，利用者数は設計値を超えていないにもかかわらず，メモリ使用量の実測基準値が設計基準値を越えているために，メモリ総消費量の実測値が設計値を超えている状況がわかる．また，メモリ使用量の実測基準値の移動平均，標準偏差も増加しており，アプリケーションに何らかの不具合が生じていると考えられる．

図 5 はシステムがキャパシティプラニングの想定通りに稼働している場合の例を示している．

5. 評価とまとめ

本研究では，キャパシティプラニングで用いた情報を実行環境での監視に活用することで，単にリソース不足の予兆を検知するだけでなくキャパシティプラニングの妥当性を検証することが可能となり，システムの信頼性を向上させる方法を示した．

関連研究として，システム統合テスト時にログ情報を収集し，性能のシミュレーションモデルを作成し，このモデルに基いて性能予測を行うという手法が提案されている(8)が，フレームワークの提案に止まっている．また，実行中のログ情報の分析により平常時の各種性能情報間の不変な関数関係を発見し，この関係が成立しない状況を検知することで，障害の発生を検知するという手法に基く製品が販売されている(9)．

これらの関連研究と本研究を比較すると，対象となる Web アプリケーションの構造がわからない場合には，実行時のログ情報から性能シミュレーションモデル，性能情報間の不変な関数関係を導くという関連研究のアプローチが有効であると考えられる．対象となる Web アプリケーションの構造がアプリケーションフレームワークの利用等により予めわかっていて，キャパシティプラニングの基礎データが存在する場合には，この情報を有効活用する本研究の手法の方が少ないデータから精度の高い結果が得られると考えられる．しかしながら，関連研究の方式の詳細が開示されていないため，詳細な比較や精度に関する定量的な評価は出来ていない．商用で用いられる Web アプリケーションは，開発効率の観点からアプリケーションフレームワークを活用することが一般的であり，この前提条件は，一般的な Web アプリケーションでは制限とはならず，本研究の結果の適用範囲が限定されることはない．

また，本研究では，ログ情報のリアルタイム監視にストリーム DB を利用する方法を確立し，プロトタイプ実装により，実用性を検証した．特に，ログ情報をストリームとして取り込む部分の工夫により，ファイルアダプタ及び，入力ストリームの数を減らし，キャパシティプラニング支援ツールでの CQL 生成を単純化する方式を示した．

6. 参考文献

(1) J. Allspaw :"The Art of Capacity Planning: Scaling Web Resources", O'Reilly Media; (2008-9), (邦訳)佐藤 直生,木下 哲也 (訳): "キャパシティプランニング ― リソースを最大限に活かすサイト分析・予測・配置"，オライリージャパン (2009-3).

(2) D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida: "Performance by Design: Computer Capacity Planning By Example", Prentice Hall (2004-1).

(3) M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia: "Above the Clouds: A Berkeley View of Cloud Computing", UC Berkeley Reliable Adaptive Distributed Systems Laboratory (http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf)(2009-2).

(4) A. Arasu, S. Babu, and J. Widom: "The CQL Continuous Query Language: Semantic Foundations and Query Execution" , Technical report, Stanford University (http://dbpubs. stanford.edu/pub/2003-67) (2003-10).

(5) C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk: "Gigascope: a stream database for network applications", Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp.647-651 (2003-6)

(6) 吉野松樹, 大場みち子，薦田憲久，山出泰子，中道繁："情報システム運用におけるストリーム DB によるメッセージ分析方式",電気学会誌 C 部門論文誌，Vol.130, No.4, pp.607-614 (2010-4).

(7) Stanford University InfoLab: "Stream Query Repository", http://infolab.stanford.edu/stream/sqr/

(8) M. Pinzger: "Automated web performance analysis, with a special focus on prediction", Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, pp.539-542 (2008-10)

(9) 加藤清志, 西村光央, 勝見順一: "WebSAM Ver.8 が実現するクラウド時代のデータセンター運用", NEC 技報, Vol.63 No.2, pp.75-78 (2010-4)