The University of Osaka
Institutional Knowledge Archive

Title	データセンタにおける情報システムの高度運用に関する研究
Author(s)	吉野, 松樹
Citation	大阪大学, 2011, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/796
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

データセンタにおける情報システムの
高度運用に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2011 年 1 月

吉野　松樹

内容梗概

情報システムは，政府・自治体業務，企業活動および一般消費者の各種の活動を支える重要なインフラストラクチャとなっており，その安定的な運用保守の費用が情報投資の 7 割に達しており，運用保守の効率化が，大きな課題となっている．情報システムのアプリケーションのライフサイクルの上流フェーズである「要件」～「展開」のフェーズの効率化の重要性は 1960 年代後半から認識され, ソフトウェア工学として研究が行われ成果が上がっている．しかしながら，ライフサイクルの最終フェーズである「運用」フェーズについては，実際に運用される現場の状況がさまざまであることから，一般化された定式化が難しく，これまで十分に研究が進んでいないのが実態である．しかし，近年コスト削減を目的として IT（Information Technology）リソースの集約を図る場合が多くデータセンタの大規模化が進んでいることや，IT リソースの運用管理を含む IT インフラストラクチャをサービスとして提供するクラウドコンピューティングの進展により,「運用」フェーズの高度化・効率化が求められるようになってきている.

このような状況に対応すべく，本研究では，大規模データセンタにおける情報システムの運用管理手順の分析を基に，ストリーム DB（Database）等の最新技術を導入し，運用管理の高度化に向け下記の研究を行う.

（1） データセンタ内の情報システムから発行される各種メッセージをストリーム DB を利用してリアルタイム分析するための手法の研究,

（2） 情報システムが出力するメッセージ/ログ情報をストリーム DB を利用してリアルタイム分析することでリソース不足の予兆検知を行う手法に関する研究,

（3） システム運用管理の観点でのデータセンタの省電力化に関する研究.

本論文は全 6 章から構成される.

第 1 章の序論では，大規模データセンタにおける情報システムの運用管理の現状を整理し，解決すべき課題をまとめる．さらに，それらの課題の解決に関して，関連する従来研究の状況を概観するとともに，本論文で提案する解決案の目的と位置づけ及びカバーする範囲について述べる.

第 2 章では, 第 3 章以降の情報システムの運用管理の自動化に向けた研究の準備として，現在実際に行われている運用管理手順を分析し，運用管理手順のパターンを抽出し，パターン記述言語による記述を行い，各パターンに対して自動化のための方針を示す.

第 3 章では，情報システムが発行するメッセージをストリーム DB を利用して分析し障

害発生を検知する手法を提案する．提案方法に基くプロトタイプを実装し実際のデータセンタでのメッセージ分析に適用し，提案手法の有効性を示す．

第 4 章では, 情報システムの IT リソース不足による障害発生の予兆を検知する方法を提案する．設計時の想定と実際の運用時の状態の差分をストリーム DB を用いて監視し，統計的検定に基いて異常状態の発生を一定の危険率に基いてリアルタイムに検定する手法を提案し，プロトタイプによる効果を検証する．

第 5 章では，情報システムの運用管理手順のパターンのデータセンタの省電力運用への適用を提案する．運用管理を高度化することが，省電力運用の高度化につながることをユースケースにより示す．さらにパターンによる分類を前提とした運用手順の記述方法を提案し，これによって省電力運用の効果の算出が容易になることを具体的事例により示す．

最後に，第 6 章では，結論として本研究で得られた成果を要約し，今後に残された課題について述べる．

研究業績

A. 学術論文誌論文

1. M. Yoshino, M. Oba, and N. Komoda: “Extending a Method of Describing System Management Operations to Energy-Saving Operations in Data Centers,” *Internatinal Journal of Computers*, Vol. 5, Issue 1, pp.115-122, 2011.

2. 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Ｃ部門論文誌, Vol.130, No.4, pp.607-614, 2010.

3. M. Yoshino, N. Komoda, and M. Oba: “An Analysis of Patterns for Automating Information System Operations,” *WSEAS Transactions on Information Science & Applications*, Vol. 5, Issue 11, pp.1618-1627, 2008.

B. 国際会議

1. M. Yoshino, N. Komoda, and M. Oba: “On a Method for Describing System Management Operations and its Use in Evaluating Energy Saving Operations, ” in *Proc. of the 9th WSEAS International Conference on Data Networks, Communications, Computers (DNCOCO '10)*, pp.47-52, 2010.

2. M. Yoshino, N. Komoda, N. Nishibe, and M. Oba: “Classification of Energy-Saving Operations from the Perspective of System Management,” in *Proc. of 8th IEEE International Conference on Industrial Informatics 2010 (INDIN2010)*, pp.651-656, 2010.

3. M. Yoshino, M. Oba, N. Komoda, T. Yamade, and S. Nakamichi: “Message Analysis Method Based on a Stream Database for Information System Management, ” in *Proc. of the 4th International Conference on Research Challenges in Information Science 2010 (RCIS2010)*, pp.519-526, 2010.

4. M. Yoshino, N. Komoda, and M. Oba: “Creating Operation Method Patterns for Automating Information System Operations,” in *Proc. of the 8th WSEAS International Conference on Applied Computer Science (ACS'08)*, pp.171-175, 2008.

5. M. Yoshino: “APL as a Prototyping Language: Case Study of a Compiler Development Project,” in *Proc. of the International Conference on APL*, pp.235 – 242, 1986.

C. 学会講演

1. 吉野松樹, 薦田憲久, 半田敦郎, 大場みち子: “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,” 第9回情報科学技術フォーラム（FIT2010） 予稿集 第4分冊, RO-001, pp.123-128, 2010.

2. 吉野松樹, 薦田憲久, 大場みち子, 西部憲和: “情報システム省電力運用のシステム運用観点での分類,” 電気学会情報システム研究会, IS-01-26, pp.135-140, 2010.

3. 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,” 電気学会情報システム研究会, IS-09-62, pp.37-42, 2009.

4. 吉野松樹, 薦田憲久, 大場みち子: “運用自動化に向けた運用手順のパターン化,” 電気学会情報システム研究会, IS-08-41, pp.35-38, 2008.

5. 千葉俊哉, 吉野松樹, 藤井啓詞, 黒井充, 荒井仁: “大規模開発支援向けOMG MOF準拠リポジトリの試作と評価,” 情報処理学会第 62 回全国大会講演集, pp. I-253 - I-254, 2001.

6. 今城哲二, 吉野松樹, 大坪稔房, 原田晃, 大野治, 植村俊亮: “オンライン業務プログラムの環境独立処理方式,” 情報処理学会研究報告（ソフトウェア工学研究会）, 2000-SE-135, pp.33-40, 2001 .

7. 末宗英利, 吉野松樹, 高舘公人: “ CSS 統合開発環境（3） :プラットフォーム,” 情報処理学会第 45 回平成 4 年後期全国大会, No.5, pp. 343-344, 1992.

8. 吉野松樹, 谷口恵子, 西山勲: “分散形第 4 世代言語 EAGLE/4GL,” 情報処理学会第 42 回（平成 3 年前期）全国大会, 5S-6, pp. 345-346, 1991 .

D. その他

1. 布広永示（編著）: “Java オブジェクト指向プログラミング,” オーム社, 2008 （第 7 章「Java 実行環境について」担当）.

2. IT トップガン育成プロジェクト著: “ソフトウェアエンジニアリング講座 3 プログラミング,” 日経 BP 社, 2007（第 6 章「システム開発におけるプログラミング」担当）.

3. 吉野松樹, 阿部欣成, 中誠一郎: “ビジネスグリッドの狙い,” 情報処理, Vol.47, No.9, pp.947-952, 2006.

4. 情報処理推進機構: “ビジネスグリッドが切り開く次世代 IT 基盤,” アスキー, 2006 （編集委員として参画）.

5. 吉野松樹: “日立製作所の「Harmonious Computing」を中心としたビジネスグリッドコンピューティングへの取り組み （特集 1 ビジネスグリッドコンピューティング）,” COMPUTER & NETWORK LAN, オーム社, Vol.22, No.8 （通巻 250 号）, pp.54-62, 2004.

6. 今城哲二監修, 吉野松樹, 湯浦克彦, 橋本恵二, 成田雅彦, 大谷真, 小池博, 高橋まゆみ, 団野博文, 銀林純, 大槻繁, 石田厚子, 増石哲也著: “ビジネスオブジェクト入門,” ソフトリサーチセンター, 2000 （第 1 章「ビジネスオブジェクトとは」, 第 2 章「ビジネスオブジェクトの技術基盤」, 第 6 章「日立製作所のアプリケーションフレームワーク」のうち 6.1 節「Network Objectplaza の概要」, 6.4 節「アプリケーションフレームワークの実例」担当）.

7. 松本吉弘, 吉野松樹, 大槻繁: “組み込みシステムのためのデザインパターンと共通利用環境,” 次世代デジタル応用基盤技術開発事業/先端的情報化推進基盤整備事業・論文集, pp.495-502, 情報処理振興事業協会, 2000.

8. トマス・J. モーブレィ （著）, ラファエル・C. マルボー （著）, 大谷真 （監訳）: “CORBA デザインパターン,” IDG コミュニケーションズ, 1998 （第 2 部「アプリケーションに関するデザインパターン」担当）.

9. 田坂光伸, 土田修己, 吉野松樹, 初田賢司: “WWW と分散オブジェクトを用いたシステムの構築を支援する WWW 連携分散オブジェクトフレームワーク,” 日立評論, Vol.80, No.5, pp.429-432, 1998.

10. 松本吉弘, 吉野松樹: “PCTE：ソフトウェアツールを移植可能にし, 共通に利用するための環境,” 情報処理, Vol.38, No.2, pp.121-128, 1997.

11. 吉野松樹, 秋山美登, 西尾高典: “分散コンピューティング環境におけるアプリケーション開発支援, ” 日立評論, Vol.76, No.6, pp.465-468, 1994.

12. 吉野松樹, 田村和敏, 稲益良夫: “ソフトウェア開発支援ツール SEWB3,EAGLE/4GL の機能と特長,” 日立評論, Vol.75, No.11, pp.727-734, 1993.

目次

第 1 章 序論 1
1.1 研究の背景 1
1.2 従来研究 4
1.3 研究の方針 5
1.4 本論文の構成 6
第 2 章 運用自動化に向けた運用手順のパターン化 9
2.1 緒言 9
2.2 運用手順のモデル化方法 9
2.2.1 情報システムの運用の活動 9
2.2.2 モデル化方法 10
2.2.3 複雑な運用手順の表現 11
2.3 運用状況の調査による運用手順のパターン化 11
2.3.1 運用手順の分析対象 11
2.3.2 作業単位の詳細分類 12
2.3.3 運用手順のパターン記述方法 13
2.3.4 運用手順のパターン抽出手順 14
2.3.5 抽出された運用手順パターン 15
2.3.6 運用手順パターンと運用目的との関係 20
2.4 運用手順パターンと運用自動化の関係 21
2.4.1 各運用手順パターンと運用自動化の関係 21
2.4.2 運用手順パターンを意識した運用自動化の検討 21
2.5 結言 22
第 3 章 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 25
3.1 緒言 25
3.2 情報システム運用におけるメッセージ分析の現状 27
3.2.1 大規模データセンタで実施されているメッセージ分析 27
3.2.2 現状のメッセージ分析方式の課題 28

3.3 ストリーム DB を適用したメッセージ分析方式 30
3.3.1 ストリーム DB の概要 30
3.3.2 ストリーム DB を適用したメッセージ分析システムの概要 31
3.3.3 メッセージ分析へのストリーム DB 適用の課題 32
3.4 メモリ使用量を意識した CQL の設計方針と自動生成方式 33
3.4.1 メモリ使用量を意識した CQL 設計方針 33
3.4.2 設計方針の評価 39
3.4.3 CQL テンプレートと自動生成方式 40
3.5 ストリーム DB によるメッセージ分析方式の適用実験 42
3.5.1 適用実験環境と適用分析パターン 42
3.5.2 分析パターン追加の容易性の評価 43
3.5.3 分析ルールのメンテナンスの容易性の評価 44
3.5.4 適用実験評価のまとめ 44
3.6 結言 45
第 4 章 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. 47
4.1 緒言 47
4.2 キャパシティプラニングの方法と問題点 48
4.2.1 キャパシティプラニングの方法 48
4.2.2 キャパシティプラニングの問題点 50
4.3 ストリーム DB 適用によるリアルタイム障害予兆検知 51
4.3.1 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 51
4.3.2 予兆検知手法 52
4.3.3 CQL 自動生成 54
4.3.4 ログ情報の取り込み 55
4.4 プロトタイプ実装による実験 56
4.5 稼働情報のリアルタイム分析への統計的手法の導入 57
4.5.1 統計的手法導入の目的 57
4.5.2 実測基準値が設計基準値と等しいことの検定 59
4.5.3 危険率の設定 61
4.5.4 検定のための CQL のウィンドウサイズ 61
4.5.5 実験の想定環境とテストデータ 62
4.5.6 実験結果 64

4.5.7 実験結果の考察 67
4.5.8 関連研究との比較 69
4.6 結言 69
第 5 章 情報システム運用手順パターンのデータセンタ省電力運用への適用拡大 71
5.1 緒言 71
5.2 システム運用管理とグリーン IT 72
5.2.1 システム管理プロセスとツールの範囲 72
5.2.2 関連する技術 73
5.3 運用手順のパターンと CMS を前提とした運用手順記述 73
5.3.1 用語の定義 73
5.3.2 運用手順記述形式 74
5.4 運用管理手順パターンに基づく電力消費削減を意識した運用手順 75
5.4.1 定期的定型運用のユースケース 75
5.4.2 不定期定型運用のユースケース 77
5.4.3 注意喚起のユースケース 78
5.4.4 障害処置のユースケース 79
5.4.5 ユースケースのまとめ 81
5.5 電力消費量削減評価と分析の例 83
5.5.1 評価で用いる事例 83
5.5.2 評価結果と分析 84
5.6 結言 87
第 6 章 結論 89
6.1 本研究のまとめ 89
6.2 今後の課題 90
謝辞 93
参考文献 95

第 1 章

序論

1.1 研究の背景

情報システムは，現代におけるあらゆる経済活動の基盤となっており，エネルギー，水，交通機関などと並ぶ社会を支えるインフラストラクチャとなっている．情報システムにかかわる諸活動を IT（Information Technology）サービスマネジメントのベストプラクティスを集めたフレームワークである ITIL（Information Technology Infrastructure Library）[(1)]では，情報システムのアプリケーションのライフサイクルを「要件」，「設計」，「構築」，「配備」，「運用」，「最適化」の 6 フェーズに分類している（図 1-1）．

図 1-1 ITIL におけるアプリケーションライフサイクル

情報システムのライフサイクルは，情報システムで実現したい内容を検討し要求仕様を定義し，必要な機能を定義し実装する開発フェーズと，実際に情報システムを稼働させる運用フェーズに大別できる. 開発フェーズに関しては, 1968 年の NATO（North Atlantic Treaty Organization）主催の会議[(2)(3)]以来,「ソフトウェア工学」として研究が進められ，その成果の開発現場での実践が進んできている．一方，運用フェーズに関しては，それぞれの運用サイト毎の環境の差異に依存する部分が大きいこともあり，開発フェーズと比較すると普遍的な議論が難しく研究が進んでいない状況であり，開発フェーズ以上に属人的かつ労働集約的に作業が行われているのが実態である.

情報システムが本来期待される価値を産み出すのは運用フェーズに入ってからであり，開発フェーズよりも運用フェーズの期間の方が長いことを考慮すると，運用フェーズにおいて情報システムの価値を引き出し，効率的に稼働させることが情報システムのライフサイクル全体での ROI（Return on Investment）向上にとって重要である．経済産業省が発行している「情報処理実態調査報告書」の 2008 年版[(4)]によると，企業における情報処理投資の内，既存システムの運用保守にかかる費用が約 7 割を占めており，新規のシステム開発への投資は残りの約 3 割となっている．IT のユーザ企業の国際競争力強化のためには，運用フェーズの効率向上を図り，競争力強化につながる新規システム開発への投資を拡大することが必要である.

近年，ユーザ企業内で各部門がそれぞれに管理・運用していたサーバ群を物理的に一箇所のデータセンタに集約するサーバ統合を進めるケースが増えている．また，集約したサーバ群の管理・運用を社外のデータセンタ事業者に委託するアウトソーシングの形態を採る場合もある．さらに，IT インフラストラクチャ，IT プラットフォームの利用をサービスとして提供する IaaS（Infrastructure as a Service）や PaaS（Platform as a Service）を利用するケースや，自社の競争優位の源泉とならないコアでない業務については，業務アプリケーションをサービスとして提供する SaaS（Software as a Service）[(5)(6)]を利用するケースも増えている．このような傾向により，自社内利用のためのデータセンタにしろ，アウトソーシング事業あるいはクラウドコンピューティング事業のためのデータセンタにしろ大規模化が進んできており[(7)]，運用管理する対象の IT リソースの数が増加している．情報システムの障害によるビジネス的な損失，社会的影響が増大しているため，データセンタでは，情報システムの信頼性を高めるために，障害の予兆検知や障害発生の発見，障害発生部位の特定，障害復旧などをユーザとの間で実施時間や実施内容を合意した SLA（Service Level Agreement）に基づいて実施することが要求されている．それと同時に，運用コスト低減と運用管理の対象となる IT リソースの数が増加しているため, 高度な運用管理を効率的に行

うための運用自動化が重要となっている．

情報システムの運用自動化については，2003 年頃から IBM 社の”Autonomic Computing”[8][9][10][11]を始めとした研究が進められている．2003 年から 3 年間実施された経済産業省の「ビジネスグリッドコンピューティングプロジェクト」[12][13][14][15][16]では，情報システムの高信頼化をコストを抑えつつ実現することが目標として掲げられており，これを実現するための中心的技術開発は，情報システムの運用自動化である．これらの成果は, IBM 社の Tivoli[1]，国内ベンダでは(株)日立製作所の JP1[2], 富士通の System Walker[3], NEC の WebSAM[4]といった各ベンダの運用管理製品で実装されており，情報システムの運用高度化・自動化に効果を上げている．しかしながら，Autonomic Computing の当初の目標と現状を比較した 2010 年の文献[17]に述べられているように，当初の目的であった自律的に稼働する情報システムを実現するには至っておらず，未ださまざまな課題が残っている．

更に, 大規模化を続けるデータセンタにおいては IT 機器ならびに付帯する空調設備等を含む消費電力の増加が大きな問題となっており，省電力運用が喫緊の課題である．データセンタの省電力化に関しては，CPU（Central Processing Unit）等のハードウェア部品レベルの研究[18]，ディスクアレイシステムにおいてアクセスのないディスクを停止させる MAID（Massive Arrays of Inactive Disks）[19]のようなアーキテクチャレベルの研究，データセンタ内の温度変化のシミュレーションによる空調の制御[20]などさまざまなレベルでの研究が行われている．

本研究では, このような状況を考慮し, 以下のような課題を解決することを目的とする．

課題（1）：情報システムの運用手順は一般的に手順書の形式でまとめられており，ノウハウの一般化ができず，運用の自動化の阻害要因にもなっている．

課題（2）：運用自動化を実現するためには，障害予兆検知，障害発生検知，障害部位特定が重要であるが，そのための有効な情報であるデータセンタ内で膨大に出力されるログ/メッセージ情報の活用が不十分．

課題（3）：情報システム運用とデータセンタの省電力運用は個別に検討されており，改善の余地がある．

---

[1] IBM Tivoli Software ホームページ: http://www-01.ibm.com/software/tivoli/
[2] 日立 JP1 ホームページ: http://www.hitachi.co.jp/Prod/comp/soft1/jp1/
[3] 富士通 Systemwalker ホームページ: http://systemwalker.fujitsu.com/jp/
[4] NEC WebSAM ホームページ: http://systemwalker.fujitsu.com/jp/

1.2 従来研究

課題（1）に関しては，情報システムの運用手順を見通しよく定義するために，運用手順のモデル化が試みられている．IBM 社の “Autonomic Computing”では，そのアーキテクチャを記述する文書[21]で，情報システムの運用手順を“MAPE”ループと呼ばれる「監視」（monitor），「分析」（analyze），「計画」（plan），「実行」（execute）フェーズの繰り返しでモデル化している．また，ビジネスグリッドコンピューティングプロジェクトでは，情報システムの運用手順を「イベント検知」「状況確認」「判断」「アクション」にフェーズ分けしモデル化している．しかしながら，具体的な個々の運用手順においてそれぞれのフェーズでどのような処理を行うのか，どのような処理の組み合わせで行われているのか，といった運用手順のデザインパターンについては分析されていない．

運用手順のモデル化の目的は，運用自動化である．運用自動化に関する関連研究[22] [23] [24]においても，運用手順のデザインパターンの分析に関しては言及されていない．メインフレームから分散システムへの移行に伴い，情報システムの複雑さが増大したことにより，運用管理の自動化に向けた AI（Artificial Intelligence）手法の導入の研究が行われている[25]．例えば，情報システムの自律運用を複数の制約条件を充足させる問題と捉え AI 手法を適用する研究[26]，性能監視に事例ベース推論を適用する研究[27]，ルールベースエージェントの提案[28]などが行われているが，実用的レベルでの適用はまだ行われていない．このような流れを受けて，標準化団体 OASIS（Organization for the Advancement of Structured Information Standards）内に 2010 年 2 月に設立された SAF（Symptoms Automation Framework）TC（Technical Committee）[5] では，情報システムが示すさまざまな兆候とその対策の関係を知識として蓄積するためのフレームワークの標準化作業が進められている．

課題（2）に関連して，データセンタにおける障害検知のためにログ情報やメッセージ情報を分析するという手法は既に実施されている [29][30]．しかしながら，分析ルールのメンテナンスを行う場合に分析ツールの再起動が必要であり，再起動中のログ情報及びメッセージ情報を漏らさず分析するために，分析ツールの運用が複雑になるという課題があることがわかっている．さらに，現状の分析ツールでは新たな分析パターンの追加に時間と工数が掛かるという課題もある．

障害の予兆検知，特に IT リソース不足の予兆検知については，アクセス数の予測が困難な一般向けの Web サービスを対象に必要な IT リソース量を，想定される負荷条件，業務

[5] OASIS SAF（Symptoms Automation Framework） TC ホームページ: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=saf.

アプリケーションのアーキテクチャから見積もるためのキャパシティプラニング手法[31][32]が提案されている．一般的にキャパシティプラニングでは，一定の負荷に対してどれだけのリソースを消費するかを設計基準値として設定し，負荷を業務要件から見積もることで，必要となるリソース量を見積もっている．予兆検知については見積もった IT リソースの利用状況を閾値監視している．キャパシティプラニングの根拠となる設計基準値の評価は, 複数のログ情報を解析する必要があることから, リアルタイムには行われていない．クラウドコンピューティング事業者がユーザに対する SLA を遵守しながら，保有する IT リソースを有効に活用するためには，より高度な予兆検知方式が必要とされる．テスト実行により性能モデルを構築し，性能を予測するという研究[33]も行われているが，フレームワークレベルの提案に止まっている．

課題（3）に関しては，最近の環境意識の高まりに伴って IT 機器及び IT 機器以外のデータセンタ設備のそれぞれについて上述のようにさまざまな省電力化の研究が行われている．データセンタ設備に限らず，一般的な商業施設等に関して，人間の快適さを維持しつつ省電力を実現するための，自動制御フレームワークの提案[34]も行われている．さらに， IT 機器とそれ以外の設備で独立に省電力運用を考えるのではなく，両者を連携させることで更なる省電力効果を目指した研究が近年始まっている．2009 年頃から発表されているデータセンタの省電力運用運用管理に関する文献 [35] [36] [37] [38]によれば, IT 機器と空調を含むデータセンタ設備を連携して運用管理することで，データセンタで消費する総電力を半減できる見込みであることが報告されている．しかしながら，現状の研究は，アドホック的アプローチに止まっており, 一般的に情報システムで利用する IT 機器の運用管理手順とそれ以外の設備機器の運用管理手順を統一的に扱う方法に関しては言及されていない．

1.3 研究の方針

本研究においては，1.1 節と 1.2 節に述べた状況を考慮し，1.1 節で掲げた課題に対し，以下のような方針で研究を行う．

課題（1）の運用手順のパターン抽出と記述方法の定式化については，ビジネスグリッドコンピューティングプロジェクトでの運用手順のモデルをベースに，実際の運用手順書を分析し運用手順のパターンを抽出する．抽出したパターンに対し，運用自動化のための指針を得ることを目標として分析を行う．さらに，パターンをベースとして運用手順を記述する方法を検討する．

課題（2）については，ログ/メッセージ情報をリアルタイム分析する必要性からストリ

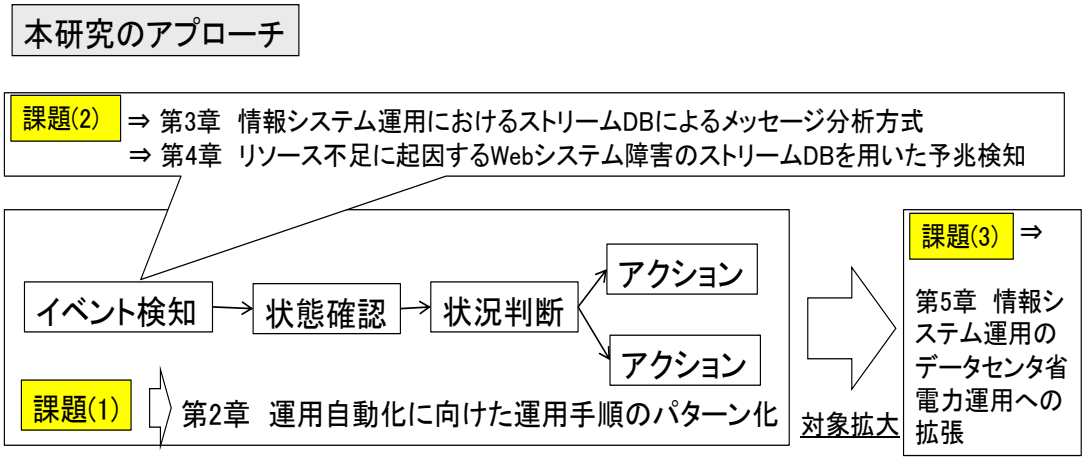


図 1-2　本研究のアプローチ

ーム DB(Database）技術の適用を検討する．ストリーム DB はリアルタイム性確保のためインメモリで全ての処理を実行することで実現しており，運用管理対象システムの規模が増大し分析すべきデータ量が増加しても分析可能となるようスケーラビリティを確保するためには，メモリ消費量を極力抑えることが適用上の課題となる．ログ/メッセージ分析処理をストリーム DB の問合わせ言語である CQL（Continuous Query Language）で記述する際に留意すべき設計方針を，メモリ消費量低減の観点で検討する．さらに，この設計方針を徹底するために CQL 自動生成方式を検討する．障害予兆検知に関しては，従来の閾値監視だけでなく，設計基準値（設計時の想定値）とおりに実際に稼働しているかを実測基準値と比較することで早期に障害の予兆を検知する手法の検討を行う．

課題（3）については，情報システムの運用手順のパターンを省電力運用という観点でとらえなおし，その有効性を確認する．データセンタ全体の設備運用管理を統一的観点でとらえることで，データセンタ全体の運用の高度化，特に省電力運用の高度化が図れるとの考えに基くものである．

それぞれの課題に対して，章を割り当て全体として図 1-2 に示すアプローチで研究を行う．

1.4 本論文の構成

本論文の構成は下記のとおりである．

第 2 章では，以降の章での情報システム運用管理の自動化に関する研究を見通しよく行

うため，文献[39][40][41]にもとづき，4 つの情報システムを題材として，実際に使われている運用管理手順書を分析し，運用管理手順のパターンを抽出する．抽出された運用管理手順のパターンに対して，そのパターンの運用手順の使用目的を分析し，さらに自動化するために留意すべき点をまとめる．

第 3 章では，文献[42][43][44]にもとづき，大規模データセンタにおいて，情報システムの障害発生を検知するために行われている，情報システムから発行されるメッセージの分析に，ストリーム DB を用いる新たな実装方式について論じる．まず，従来から大規模データセンタで使用されているメッセージ分析ツールの運用面および機能面での課題を明らかにし，その課題を解決するために，近年，時系列データ系列のリアルタイム分析に有効であるとして注目されているストリーム DB を適用する手法を提案する．さらにストリーム DB を大規模に適用する場合に課題となるストリーム DB のメモリ消費量削減のための解決方法として，CQL 設計方針を提案する．提案した CQL 設計方針を強制することと，CQL 作成・修正の効率向上を目的として CQL 自動生成方式を提案する．提案方法に基くプロトタイプを実装し実際のデータセンタでのメッセージ分析に適用し，提案手法の有効性を示す．

第 4 章では，文献[45][46]にもとづき，情報システムが利用者数の増加等によって IT リソースが不足し，障害発生に至る前にその予兆を検知する方法について提案する．情報システムが出力するメッセージ，ログ情報をストリーム DB でリアルタイムに分析することで，システム稼動中の設計基準値相当の値である実測基準値を求める．実測基準値の設計基準値からの偏差を，閾値監視，線形外挿，統計的検定で評価することで，キャパシティプラニング時の想定とおりに情報システムが稼働しているかどうかを検証する．これにより，単にリソース消費量を閾値監視する方式と比較して早期にリソース不足の予兆を検知する方式を提案する．また，統計的検定を用いた稼働監視において，ストリーム DB のメモリ消費量削減の観点から，適切な CQL の設計指針を示す．

第 5 章では，文献[47][48][49][50]にもとづき，第 2 章で分類した情報システムの運用管理手順のパターンを活用することでデータセンタの省電力を図る考え方について述べる．第 2 章で抽出した運用手順のパターンに基く運用手順の記述方法を提案し，それを用いてデータセンタ全体の運用手順を記述できることを示す．さらに，この記述方式を用いて，運用管理を高度化することで省電力を図る方法，および逆に省電力を推進するためにはどのように運用管理環境を整備すべきかということを事例を交えて論じる．

最後に，第 6 章では，結論として本研究で得られた成果を要約し，今後に残された課題について述べる．

第 2 章

運用自動化に向けた運用手順のパターン化

2.1 緒言

本章では，以降の章での情報システム運用管理の自動化に関する研究を見通しよく行うために，運用管理手順のパターンの抽出について論じる．

情報システムの運用コスト削減には，運用手順の自動化が効果的であり，システム運用管理の自動化を目的としたツールがベンダ各社から提案され，商用の製品も提供されているが，現実の運用自動化の状況はまだ不十分である．その原因の一つとして運用自動化に適した運用手順の設計手法が確立していないことが挙げられる．ソフトウェアの設計・開発におけるパターン化の有効性は, Gamma らによるオブジェクト指向設計におけるデザインパターン(51)によって示され，その後さまざまな分野においてデザインパターンの考え方が導入されている(52) (53) (54)．デザインパターンの考え方は，情報システムの運用手順の設計においても有効であると考えられる．

そこで，本章では，運用手順をモデル化する方法を提案し，実際の情報システムで行われている運用手順を分析し運用手順のパターンを抽出する．抽出したパターンに対して自動化の方針を検討することで，個々の運用手順毎に自動化を検討する場合と比較して，効率的に自動化の検討を行えることを示す．

本章の構成は，以下のとおりである．2.2 節で，ビジネスグリッドコンピューティングプロジェクトでの検討の結果をベースとした運用手順のモデルを示す. 2.3 節で 4 つの実際の情報システムの運用手順書の分析に基くパターンを示す．運用手順のパターンを記述するため記述方法も合わせて説明する．2.4 節では 2.3 節で抽出した各パターンと運用手順の自動化との関係について論じる．

2.2 運用手順のモデル化方法

2.2.1 情報システムの運用の活動

情報システムの運用は，情報システムが持続的にその目的とする機能を一定のサービスレベルを保って提供できるようにするための一連の活動である．情報システムの運用には次のような活動が含まれる．

（1）　システム及びその構成要素の起動，終了，部分閉塞，再開.

（2）　システムの構成変更・改修（バージョンアップ）.

（3）　システム及びその構成要素の動作状態を確認するための情報(各種メトリックス)の取得とその評価.

（4）　システム及びその構成要素に障害が発生した場合の対応.

（5）　障害発生にデータ復旧するためのデータの定期的なバックアップ.

これらの活動は，運用設計で検討され運用手順として具体化される．運用手順は，運用手順書として文書として用意されてオペレータによって実行される場合，シェルスクリプトなどで実装されて実行される場合，運用自動化ツールのスクリプトとして表現されて実行される場合などがある.

2.2.2 モデル化方法

上記のいずれの場合でも，運用手順は，ワークフローとして定義することができる.

2003 年度から 2005 年度まで実施された経済産業省のビジネスグリッドコンピューティングプロジェクトにおいて，データセンタの運用自動化のために運用手順をワークフローで表現する検討[55]が行われている．本節では，その結果を参考にして，運用手順を以下のようにモデル化する.

Ⅰ. 運用手順は，作業単位をノードとする有向非循環グラフである．エッジの向きは作業単位実行の順序を表す.

Ⅱ. 作業単位は以下の 4 種類に分類される.

イベント検知（E）: 運用手順の初期状態であり，予め定義された運用管理対象の状態変化などを検知して次の作業単位に遷移する.

状態確認（C）: 予め定義された運用管理対象に対し，予め定義された状態確認を行い確認結果を引渡して次の作業単位に遷移する.

判断（J）: 状態確認から引き渡された確認結果により，次に遷移する作業単位の選択を行う.

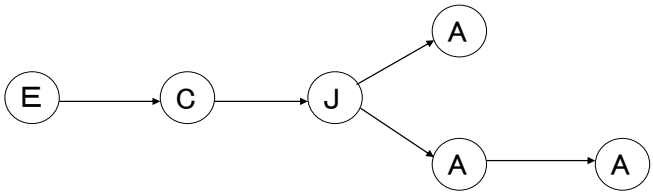
アクション（A）: 予め定義された運用管理対象に対し予め定義された操作を行う.

Ⅲ. グラフの構造について下記の制約を課す

（1）グラフのルートは，単一のイベント検知ノードである.

（2）グラフのリーフはアクションノードである.

（3）判断ノードには状態確認ノードを起点とするエッジが存在しなければならない.



Ｅ：イベント検知：運用手順の開始のきっかけとなる状態変化の検知.
Ｃ：状態確認：運用手順に関係する対象の状態確認.
Ｊ：状況判断：状態確認の結果による次に実行する作業単位の選択.
Ａ：アクション：対象に対する操作.

図 2-1　運用手順の例

（4）判断ノードを起点とするエッジの終点はアクションノードあるいは状態確認ノードである.

この定義に基づく運用手順の例を図 2- 1 に示す.

2.2.3 複雑な運用手順の表現

このモデルに従った運用手順を組合わせることで，データセンタ運用で実際に行われている複雑な運用手順を表現することが可能である．モデルに従った運用手順を組合わせるためには，ある運用手順の基本要素の「アクション」としてイベントの発行を行い，それを別の運用手順の基本要素の「イベント検知」で検知するという形で運用手順の連鎖で表現すればよい．運用手順の基本要素はループ構造を含まない非循環グラフであるが，この連鎖を用いることで，ループ構造を持つ複雑な運用手順でも表現することができる．運用手順の自動化を考える際には，基本となるモデルに従った運用手順の自動化を検討しておけば，運用手順の連鎖により複雑な運用手順の自動化も行える．自動化の方針を見通しよく行うために，次節で運用手順の基本要素のパターンの抽出を行う.

2.3 運用状況の調査による運用手順のパターン化

2.3.1 運用手順の分析対象

運用手順のパターン抽出のため，表 2-1 に示す 4 つの情報システムの運用手順書を分析し運用手順を整理した.

表 2-1　分析対象システム

	システム概要	システム規模	手順数
事例 1	Web による商品販売	Web サーバ 14 台，AP サーバ 4 台，DB サーバ 4 台	29
事例 2	携帯電話からの写真投稿サイト	Web サーバ 3 台，DB サーバ 2 台	17
事例 3	J2EE サーバを利用した Web システム	Web サーバ，AP サーバ，DB サーバからなる Web 3 階層システムを想定	21
事例 4	J2EE サーバを利用した電子帳票システム		47

事例 1 と事例 2 は実際にインタネット上で一般向けにサービスを提供中のシステムの運用手順書を分析対象としている. 事例 3 と事例 4 は実際に稼動しているシステムではなく，J2EE（Java 2 Enterprise Edition）アプリケーションサーバあるいは電子帳票ワークフロー製品を利用して情報システムを構築する際に，製品が提供する機能を有効活用するための運用管理ベストプラクティスをまとめたモデル運用管理手順書を分析対象としている．

対象とした事例はいずれも現在情報システムの主流である J2EE で構築された Web ベースシステムである．いずれの事例とも複数台のサーバから構成されており，継続的な信頼性確保と性能維持のためにきちんとした運用管理が重要となるシステムである．事例 1 と事例 2 は，インタネットで不特定多数にサービスを提供するシステムであり，サービス停止によるビジネスへの影響が大きい．そのため，運用手順には基本的な起動・終了，バックアップなどの他アプリケーションのバージョンアップ，セキュリティ対策のための保守運用，トラフィック増大時の対応，障害発生時の対応などが含まれている．事例 3 と事例 4 は, J2EE アプリケーションサーバあるいは電子帳票ワークフロー製品の機能を活用した，起動･終了，バックアップ取得，バージョンアップ作業，障害発生時の対応などの手順が含まれている．

2.3.2 作業単位の詳細分類

運用手順を，ワークフローととらえパターンによる分類を考える．ワークフローのパターンの分類として Workflow Patterns Initiative のパターン集[56]がある．これは，ワークフローの制御構造に注目したパターンである．運用自動化のためには，運用手順を構成する作業単位を自動化できるかどうかがポイントとなる．そのため，運用手順のパターン抽出にあたっては，運用手順のワークフローとしての制御構造だけでなく，作業単位の分類を考慮したパターン抽出が必要である．

運用手順の構成要素である作業単位のうち，イベント検知，アクションについて，表 2-1 に示す事例における運用手順を解析した結果，以下のような分類に分けられることがわかった．

（1） イベント検知の詳細分類：

タイマ監視：予め定められた時刻の到達を検知して次の作業単位に遷移する．

能動的オペレータ操作：オペレータの判断に基く能動的な操作が行われたことを検知して次の作業単位に遷移する．

イベント監視：予め登録されているイベントの発生を監視し，その発生の検知して次の作業単位に遷移する．

イベント監視の具体的な例としては，監視対象の状態変化，特定のメッセージ出力，メトリックスの閾値越え等が挙げられる．

（2） アクションの詳細分類：

注意喚起：オペレータに対してメール送信，ブザー鳴動，パトランプ点燈などの方法で注意喚起を行う．

データ操作：状態復旧，状況調査等の目的で予め指定されたデータの退避，保存，削除などの操作を行う．

処理実行：システムの実行を継続させるために必要な処理の実行.

2.3.3 運用手順のパターン記述方法

運用手順のパターンを抽出する際に，抽出したパターンをどのような属性を用いて記述するかが，抽出されたパターンをうまく活用する上でポイントとなる．適用分野は異なるが，Gamma らによるオブジェクト指向設計におけるデザインパターンカタログ[51]におけるパターン記述言語は大きな成功を収めた例である．これを参考にして，運用手順のパターン記述方法を検討する．Gamma らは，デザインパターンを以下のような 13 の属性を用いて記述している．

（1）パターン名称と分類，（2）パターンの意図，（3）別名，（4）動機，（5）適用対象，（6）構造，（7）登場人物，（8）協調関係，（9）効果，（10）実装例，（11）サンプルコード，（12）適用実例，（13）関連するパターン．

13 の属性の中で，（3）別名，（4）動機，（7）登場人物，（8）協調関係，（10）実装例，（11）サンプルコード，（13）関連するパターンの 7 つの属性は，運用手順のパターン記述においては不要であると判断した．その判断の詳細は，以下のとおりである．運用手順のパターンとして既に知られたものはないので，（3）別名は不要．運用手順のパターン全体が最終

的に自動運用を目的としたものであり，個々のパターンについては（4）動機で記述すべき内容は特にないため（4）動機は不要．（7）登場人物は，運用手順のパターンにおいては作業単位だけであり明白であるため不要．運用手順においては作業単位間の関係は明白であるため，（8）協調関係は不要．運用手順はプログラムではないため，（10）実装例，（11）サンプルコードは不要．運用手順のパターン抽出においては，他の運用手順との関連については分析を行わないため，（13）関連するパターンの属性は不要.

不要と判断した属性を除いた（1）パターン名称と分類，（2）パターンの意図，（5）適用対象，（6）構造，（12）適用実例，に対して運用手順のパターンであることを考慮してそれぞれ属性の名称を以下のように見直した．（1）パターン名称と分類→カテゴリ/サブカテゴリ，（2）パターンの意図→概要，（5）適用対象→運用目的，（6）構造→グラフ表現，（12）適用実例→適用例.

上記の名称の見直し及び，記述順序の調整を行った結果に対し，自動化に関する考察，という属性を追加し，以下の 6 つの属性で運用手順のパターンを記述する.

（1）カテゴリ/サブカテゴリ，（2）概要，（3）グラフ表現，（4）運用目的，（5）適用例，（6）自動化に関する考察

自動化に関する考察を追加するのは，運用手順のパターン抽出の目的は各パターンの属する運用手順の自動化であり，そのためにパターン毎に自動化するための条件をまとめておくことが有効だからである.

2.3.4 運用手順のパターン抽出手順

表 2-1 に示した 4 つの事例の運用手順を整理した結果，運用手順のグラフの形状としては，「イベント検知」と「アクション」からなるもの，「イベント検知」「状態確認」「判断」「アクション」からなるものの 2 種類があることがわかった．次に，「イベント検知」「アクション」の詳細分類を考慮して分類を細分化を行った.「イベント検知」と「アクション」からなる運用手順においては，「イベント検知」が「タイマ監視」あるいはそれ以外かによって，定期的に実施される運用手順かどうかで分類することができ，さらに「アクション」が「注意喚起」かそれ以外かで，運用オペレータに対する注意喚起だけを目的とした運用手順か，具体的な処理を伴う運用手順なのかに細分化できる．細分化により，組み合わせの可能性としては，2×2=4 通り考えられるが，「イベント検知」が「タイマ監視」であって，「アクション」が「注意喚起」であるような運用手順の事例は分析した範囲には存在せず，この組み合わせは単に特定の時刻になったことを運用オペレータに連絡するだけとなり，運用手順として意味がないため，パターンとして抽出するのは 3 通りとなる.

次に，「イベント検知」「状態確認」「判断」「アクション」からなる運用手順に対しても作業単位の詳細分類による細分化を行った．ただし，この場合には「イベント検知」が「タイマ監視」であるケースはなく，「イベント検知」では細分化の余地はない．この場合には「判断」の結果によって，何も処理を行わないケースを含めて複数の「アクション」が存在する．「アクション」が「注意喚起」だけか，それ以外の処理を含むかで，さきほどの場合と同様に運用オペレータに対する注意喚起だけを目的とした運用手順か，具体的な処理を伴う運用手順なのかに細分化できる．この場合には 2 通りのパターンが抽出される．

このうち，具体的な処理を伴う運用手順パターンを運用目的の観点で分析すると，障害運用で使用する運用手順と，通常運用あるいは保守運用で使用する運用手順の 2 つに更に細分化できることがわかった．

ここで，運用目的は以下に示す 3 つに分類している．

（1）通常運用：通常運用の中で実施される手順.

（2）保守運用：構成変更，バージョンアップ等を契機に実施される手順.

（3）障害運用：障害あるいは障害の可能性がある事象が発生した時に実施される手順.

よって，「イベント検知」「状態確認」「判断」「アクション」からなる運用手順については 3 つのパターンが抽出されることになり，全部で 6 つのパターンが抽出される．運用手順のパターンを特徴づけるのは，「イベント検知」と「アクション」の詳細分類と運用目的であるので，上記で抽出した 6 つのパターンの中で，「状態確認」「判断」の存在の有無を小分類とし，その部分だけが異なるものは同一の大分類としてまとめる．大分類に対して，以下のような名称を付与する．「イベント検知」が「タイマ監視」である大分類は，「定期的定形運用」．運用目的が通常運用あるいは保守運用であり，「注意喚起」ではない「アクション」を持つパターンの大分類は，「不定期定形運用」．「アクション」が「注意喚起」のみであるパターンの大分類は，「注意喚起」．運用目的が障害運用であるパターンの大分類は，「障害処置」とする．表 2-2 に，上記分析の結果得られた運用手順のパターンの一覧を示す．

2.3.5 抽出された運用手順パターン

表 2-2 に示した各パターンについて 2.3.3 節で示した属性を用いて記述する．

（1） パターン１：

カテゴリ：定期的定形運用

表 2-2　運用手順パターン

大分類	小分類	パターン番号
定期的定形運用	－	パターン 1
不定期定形運用	状態確認なし	パターン 2－1
	状態確認あり	パターン 2－2
注意喚起	状態確認なし	パターン 3－1
	状態確認あり	パターン 3－2
障害処置	－	パターン 4

概要：本パターンは，一定時刻に処理実行（起動，停止，再起動など），あるいはデータ操作を行う場合に用いられる．

グラフ表現：図 2-2 参照．図中の作業単位の下の記述は，2.3.2 節で示した作業単位の詳細分類である．

運用目的：通常運用

適用例：日々のシステム起動，終了操作，DB バックアップ操作等．

自動化に関する考察：自動運用されている場合が多く，自動化されていない場合でも自動化することは容易である．イベント検知は，タイマイベントによる自動化可能．アクションの自動化は，アクションの作業単位をオペレータの介入を必要としないように実装できるかどうかに依存する．

（2）　パターン 2－1：

カテゴリ：不定期定形運用．サブカテゴリ：状態確認なし

概要：本パターンは定形運用であるが，実施するタイミングに任意性がある場合や，別の作業の終了を確認した後実施する必要がある場合に用いられる．イベント監視の結果

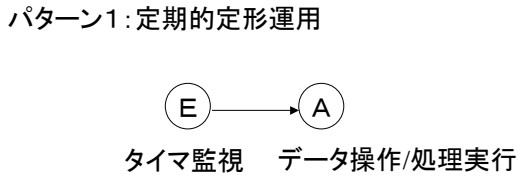


図 2-2　パターン 1：定期的定形運用

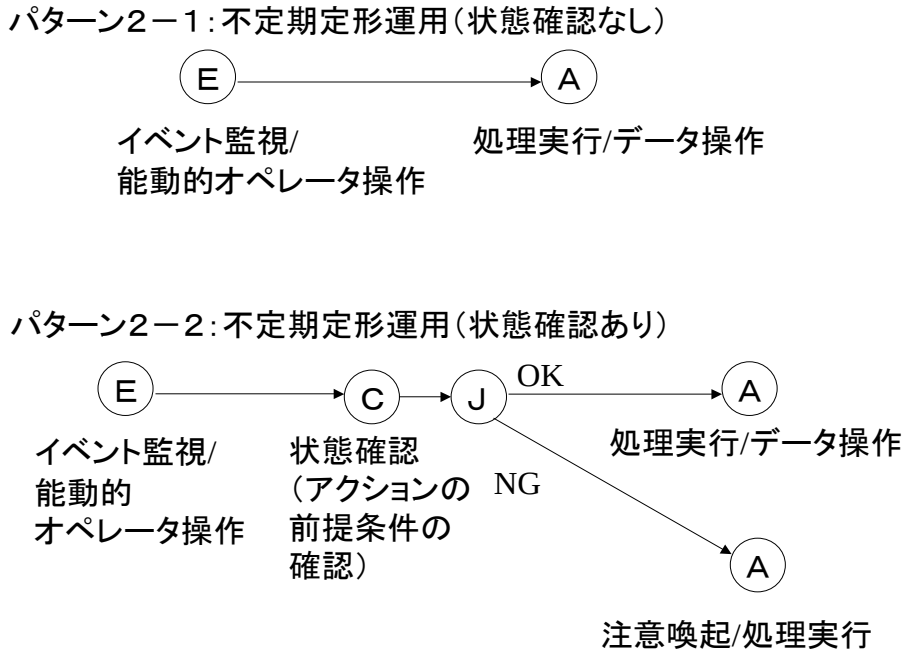


図 2-3　パターン 2：不定期定形運用

あるいはオペレータの能動的操作によりアクションが実行される．

グラフ表現：図 2-3 参照

運用目的：通常運用あるいは保守運用

適用例：オペレータ判断によるバックアップ取得，ソフトウェアアップデート，バージョン/リシビジョンアップ作業など

自動化に関する考察：イベント検知がオペレータの能動的操作の場合，イベント監視とし，アクションを自動化可能とすれば自動化が可能である．

（3）　パターン 2－2：

カテゴリ：不定期定形運用．サブカテゴリ：状態確認あり

概要：本パターンは定形運用であるが，実施するタイミングに任意性がある場合や，別の作業の終了を確認した後実施する必要がある場合で，アクションの実施のための前提条件が成立していることの確認が必要な場合に用いられる．前提条件が成立していない場合はシステムの状態の不整合として注意喚起を行う．

グラフ表現：図 2-3 参照

運用目的：通常運用あるいは保守運用

適用例：オペレータ判断によるデータベース再編成作業において，データベースにアクセスするフロントエンドエンドシステムの停止を確認してからデータベース再編成作業を実施する．

自動化に関する考察：パターン 2－1 の条件に加え，状態確認の自動化が条件となる．

（4） パターン 3－1：

カテゴリ：注意喚起．サブカテゴリ：状態確認なし．

概要：監視対象の状態の変化をオペレータ等に連絡して，必要に応じて適切な処置を行うよう注意喚起する場合に用いる．

グラフ表現：図 2-4 参照．

運用目的：通常運用，保守運用，あるいは障害運用．

適用例：不具合が発生したことをオペレータに通知する．

自動化に関する考察：イベント監視と注意喚起を自動化することで自動化可能である．両者とも自動化は容易であり，本パターンの運用手順は自動化されている場合が多い．

備考：注意喚起後にオペレータ操作が必要な場合，その操作が自動化可能かどうかを検討することが，運用手順全体の自動化の観点で必要である．

（5） パターン 3－2：

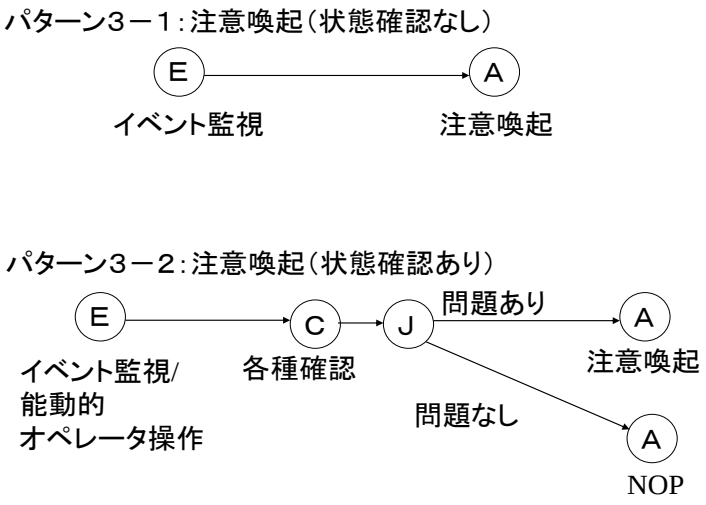


図 2-4　パターン 3：注意喚起

カテゴリ：注意喚起．サブカテゴリ：状態確認あり．

概要：イベント監視あるいは能動的オペレータ操作によって運用手順が開始され，状態確認を行って問題がある場合に注意喚起を行い，問題がなければ何もしないという場合に用いる．

グラフ表現：図 2-4 参照．

運用目的：通常運用，保守運用，障害運用

適用例：オペレータの判断に基いて，情報システムの状態を確認し，問題があれば注意喚起する．

自動化に関する考察：自動化のためには，イベント検知が能動的オペレータ操作の場合には，イベント監視に変更する必要がある．状態確認の自動化も必要である．

（6） パターン 4

カテゴリ：障害回復

概要：本パターンは，障害発生時の回復運用に用いられる．監視対象の状態の変化を契機として，アクションを実行するための前提条件の確認を行った後，前提条件が成立していれば予め定められたアクションを実行する．前提条件が成立していない場合には，注意喚起を行う．

グラフ表現：図 2－5 参照．

運用目的：障害運用

適用例：障害発生時に原因調査に必要なログデータを退避し，その後システムの再立ち上げ操作を行う．

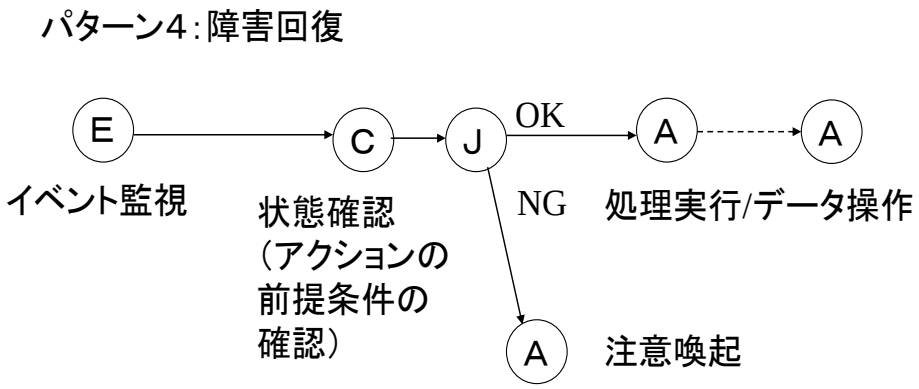


図 2-5　パターン 4：障害回復

自動化に関する考察：本パターンの運用手順については，イベント検知，状態確認，アクションの作業単位が自動化可能かどうかに依存する．

2.3.6 運用手順パターンと運用目的との関係

表 2-1 に掲げた事例の手順に対して，上記の運用目的と運用手順パターンとの関係を分析した結果を表 2-3 に示す．2.3.5 節に示した各パターンの「運用目的」属性は，この分析の結果をもとに記述している．

定期的定形パターン（パターン 1）は，全て通常運用で用いられている．

不定期定形パターン（パターン 2）は，通常運用または保守運用で用いられている．保守運用は定期的に行われるものではなく，バージョンアップ等を契機に行われる場合が多いことに呼応している．

注意喚起パターン（パターン 3）は，状態確認なし（パターン 3-1）は障害運用での使用が多く，状態確認あり（パターン 3-2）では保守運用での使用が多い．これは，注意喚起（状態確認なし）は，常時監視対象を監視し問題が発生した場合に注意喚起を行うという利用が多く，注意喚起（状態確認あり）は，バージョンアップ等の保守作業の中で作業手順の確認に用いられる場合が多いためであると考えられる．通常運用でも利用されている場合があり，運用目的によらず利用されている．

障害回復パターン（パターン 4）は，全て障害運用での利用である．

表 2-3　運用手順パターンと運用目的の関係

	通常運用					保守運用					障害運用				
	事例1	事例2	事例3	事例4	合計	事例1	事例2	事例3	事例4	合計	事例1	事例2	事例3	事例4	合計
パターン1	5	0	0	1	6	0	0	0	0	0	0	0	0	0	0
パターン2－1	0	2	2	9	13	0	8	7	24	39	0	0	0	0	0
パターン2－2	0	0	7	0	7	10	0	1	1	12	0	0	0	0	0
パターン3－1	0	0	0	0	0	0	0	0	0	0	6	3	1	0	10
パターン3－2	0	0	0	1	1	0	0	1	7	8	0	1	0	0	1
パターン4	0	0	0	0	0	0	0	0	0	0	8	3	2	4	17

2.4 運用手順パターンと運用自動化の関係

2.4.1 各運用手順パターンと運用自動化の関係

各パターン毎に，運用自動化の観点から考察をまとめる．

パターン 1 は，容易に自動化可能である．UNIX の cron や，ジョブスケジューリングツールなどを利用して自動化が行われている．

パターン 2 は，「イベント検知」が「能動的オペレータ操作」のままではオペレータ介入が必要となるので，自動化のためには「能動的オベレータ操作」を「イベント監視」に置換える必要がある．パターン 2-2 の場合には，さらに「状態確認」及び「判断」の作業単位がオペレータの介入なしに自動化できる必要がある．

パターン 3－1 は，自動化可能である．ただし「アクション」が「注意喚起」であるため，具体的な対処が必要な場合にはオペレータ介入が必要となりそこの部分は自動化できない．その部分まで含めた自動化を行うには，具体的対処の「アクション」を自動化し，パターン 4 の運用手順にする必要がある．

パターン 3－2 は，「イベント検知」が「能動的オペレータ操作」の場合には，自動化するためには，「イベント監視」に変更する必要がある．「状態確認」及び「判断」の作業単位もオペレータの介入なしに自動化できる必要がある．「アクション」については，パターン 3－1 と同様である．

パターン 4 を自動化するためには，「イベント検知」「状態確認」「判断」「アクション」の全てを自動化する必要がある．

2.4.2 運用手順パターンを意識した運用自動化の検討

情報システムの運用手順の自動化を検討する際に，個々の運用手順に対して個別に自動化を検討するのでは効率が悪い．運用手順のパターンを利用して次のような手順で運用自動化を検討することで，効率的に検討を進めることができる．

（1）運用手順をパターンに分類する．この際，複雑な運用手順は，2.2.3 節に述べた運用手順の連鎖に分解した結果を分類する．

（2）自動化できている運用手順と自動化できていない運用手順に分類する．

（3）自動化できている作業単位をリストアップする．

（4）自動化できていない運用手順に対して，2.4.1 節に記述した内容に基いて，自動化するために必要な作業をリストアップする．

（5）自動化できていない運用手順に対して，自動化の優先順位付けを行う．

（6）自動化の優先順位と，（4）でリストアップした自動化に必要な作業の作業量を勘案して自動化の計画を立案する．

（6）の自動化のために必要な作業量見積りの際に，（3）でリストアップした自動化済み作業単位の中に利用できるものや参考になるものがないかを検討することで効率化を図ることができる．例えば，自動化済みの作業単位が，何らかの運用管理ツールを利用して自動化を実現している場合に，自動化できていない作業単位の自動化にそのツールが利用できる可能性がある．あるいは，新たに市販の運用管理ツールを導入，あるいはツールを開発することによって（4）でリストアップした自動化に必要な作業のいくつかをまとめて自動化できる場合もある．

作業単位種別でみると，「イベント検知」を自動化することで「注意喚起」パターンの運用手順の自動化が図れるが，「アクション」が自動化されても「イベント検知」が自動化できないと「障害回復」パターンの自動化はできない．運用手順のパターン化により，このような観点も考慮して自動化の検討を行うことができる．

運用手順のパターン分類を活用し，上記のような手順で運用手順の自動化を検討することで，個々の運用手順毎に個別に自動化を検討するよりも，見通しよく自動化を検討することができる．

2.5 結言

運用手順の自動化の検討を効率良く行なうために，運用手順のモデルをベースに実際の情報システムで用いられている運用手順を運用手順書を分析することで整理し，４種類の運用手順のパターンを抽出した．運用手順向けのパターン記述形式を提案し，抽出したパターンはそれを用いて記述し，記述形式の記述能力を確認した．運用手順のパターンと運用目的との関係を分析し，運用目的に応じて適切な運用手順パターンを選択するための指針を得た．さらに運用手順の自動化の検討を，各パターン単位で行うことで，個々の運用手順毎に行うよりも効率的に行う手順を示した．

本章で抽出した運用手順のパターンは，現在，系統的な確立した方法論が存在しない運用設計の方法論を検討する際の出発点となると考える．また，現在，ベンダ毎に独自の形式となっている運用手順自動化ツールの定義形式の標準化を検討する場合のベースとなりうる．

本論文の第 3 章，第 4 章，第 5 章では，本章の検討結果を前提として議論を進める．

本章の研究成果を活用して，ソフトウェア開発における設計・開発方法論に匹敵する方

法論を運用に関しても構築していくことと，その方法論を実装する運用手順自動化ツールの充実を図ることが，運用の高度化，コスト低減のための課題である．

第 3 章
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

3.1 緒言

本章では，システム運用管理の高度化の中で，運用手順開始のきっかけを与える「イベント検知」の高度化，特にデータセンタ内で多量に発行されるメッセージを分析することで障害発生を検知する手法について論じる．

第 2 章で抽出した運用手順のパターンを自動化，高度化するためには，運用手順開始のきっかけとなるイベント検知の自動化，高度化が必要である．イベント検知の手段の一つとして，運用監視の対象となる情報システムが出力する各種メッセージを監視する方法がある．大規模なデータセンタでは，ある特定のメッセージが出力されたら，定められた運用手順を実行するといった単純なメッセージの監視でなく，メッセージの発生頻度，発生順序などの条件も考慮したメッセージ分析によって，正確に情報システムの状態を把握することが行われている．

大規模データセンタ内で大量に発行されるメッセージを関連付けてリアルタイムに分析するというように，大量かつ複雑な関係を持ったイベント間の関係をリアルタイムに分析するという課題を解決するための技術は，一般的に CEP（Complex Event Processing）[57] と呼ばれ，研究およびその成果を活用した製品が市場に出始めている．

著者らがヒアリング調査を実施したデータセンタでは，運用管理者が過去の障害事例をベースにメッセージ種別，メッセージの発行元などを特定したメッセージの出現パターンを事例データベースに登録し，登録されたパターンに合致するメッセージの出現を検知するメッセージ分析ツールを開発している．しかし，メッセージ出現パターンごとにソフトウェアロジックを用意する必要があり，新たなメッセージの出現パターンへの対応の工数が掛かること，事例データベースのメンテナンスのためにメッセージ分析ツールを再起動する必要があることなど，煩雑な運用が問題になっている．

このような問題点を解決するために，現状，専用の分析ツールを開発して適用しているデータセンタ運用におけるメッセージ分析に，CEP 技術の適用を検討する．CEP 技術の実

現方式には，独自のルール言語を用いるもの，CEP を実現するための API を提供するものなどさまざまあるが，本研究においては CEP 技術の一つであるストリーム DB の適用を検討する．ストリーム DB を選択した理由は，CEP 処理の記述に，産業界で広く普及している RDB（Relational Database）に対するデータベース問合わせ言語である 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 [58]を利用するため，実適用時の初期導入が容易であることと，CQL で宣言的に処理を記述するため処理内容の変更がプログラムのリコンパイルなどの操作を行わずに容易に行えるため，柔軟性を要求される実際の適用に適していると判断したからである．

ストリーム DB は，CEP 技術の一つであり，近年注目され活発に研究が進められている．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， CQL によって問合わせを行う．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 などの事例が報告されている[59]．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[6]に示されている．

ストリーム DB をデータセンタにおけるメッセージ分析に適用する場合には以下のような課題がある．適切な CQL の記述は熟練とノウハウが必要であり，データセンタの運用管理者が自ら CQL を記述することは困難である．特に，ストリーム DB は問合わせの処理をリアルタイムで行うためにメモリ上で全ての処理を行う実装方式を採っており，性能を確保するためにはメモリ使用量を考慮した CQL の設計が必要である．

そこで，上記の課題を解決するために，ストリーム DB を適用することで専用のソフトウェアの開発を不要とするメッセージ分析方式を提案する．さらに，性能を意識した CQL の設計方法と，それに基づく CQL の自動生成方式を提案する．

本章の構成は，以下のとおりである．まず，3.2 節で現状のメッセージ分析ツールの問題点を述べる．3.3 節で 3.2 節の問題点を解決するためにストリーム DB の適用を提案する．3.4 節でストリーム DB を適用する上での課題であるメモリ使用量削減のための，CQL 設計方針及び設計方針に基く CQL を生成するための生成方式について述べる．3.5 節で提案方式を実際に大規模データセンタで適用した実験により，提案方式の効果を検証する．

---

[6] Stanford University InfoLab: “*Stream Query Repository*,” http://infolab.stanford.edu/stream/sqr/

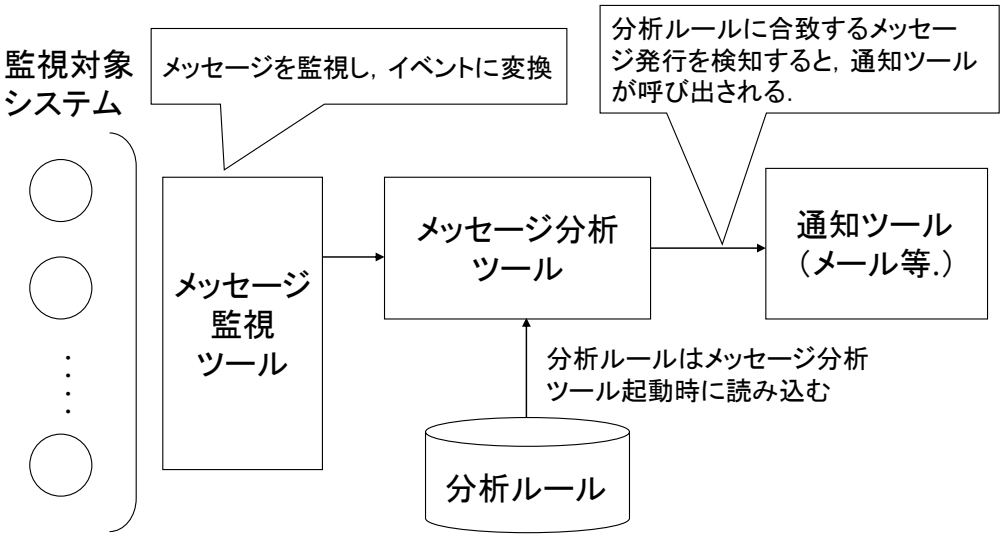


図 3-1　現状のメッセージ分析ツールの概要

3.2 情報システム運用におけるメッセージ分析の現状

3.2.1 大規模データセンタで実施されているメッセージ分析

筆者らが行った大規模データセンタの運用のヒアリング調査によれば，運用手順を開始するきっかけを与えるイベント監視の一つの形態として，以下に示すようなメッセージ分析が実施されている．

準備として，過去にシステム障害が発生した時に監視対象システムが出力していたメッセージの出現状況を調査し，分析ルールとしてリストアップする．例えば，ある特定のメッセージが 1 分間に 5 回以上発行されたといったものが分析ルールである．分析ルールを一般化し分析パターンとして定義する．例えば，一定時間内に同一のメッセージが一定回数発生するといったものが分析パターンとなる．分析ルールを一般化したものが分析パターンであり，分析パターンに具体的なパラメタを与えたものが分析ルールである．分析パターンごとにメッセージ分析ツールに検知ロジックを用意し，個々の分析ルールの情報はこの検知ロジックのパラメタとして与えられる．分析ルールと合わせて，分析ルールで定義された状況の発生が検知されたとき実行すべきアクションの情報を登録する．

実際の運用監視時には，メッセージ分析ツールが監視対象システムから発行されるメッセージを分析し，登録されている分析ルールに合致するメッセージ発行を検知すると登録されているアクションを実行する．

表 3-1 分析パターンとルールに占める割合

項番	分析パターン	構成比
1	特定メッセージの発行	70%
2	指定された時間内に指定された複数回を超える同一メッセージ発行	25%
3	指定された時間内に予め決められている全てのメッセージの発行	4%
4	その他	1%

上記で説明した，メッセージ分析ツールの構成を図 3- 1 に示す．監視対象システムのメッセージ監視は，ベンダから提供されているシステム運用管理製品を用いている．ベンダが提供するシステム運用管理製品では多くの場合，監視対象のメッセージが発行されたことを検知するとイベントとして通知する機能を提供している．メッセージ分析ツールは，このイベント通知を入力として分析を行い，障害として通知すべきパターンに合致するメッセージ発行イベントを発見すると，予め定められた方式（メール送信，パトランプ鳴動など）によって注意喚起を行う．このようなメッセージ分析は第 2 章の運用手順のパターン分析結果で挙げた「注意喚起パターン」の一例である．このパターンでは，イベント検知によって障害発生あるいは障害発生の可能性を検知すると，オペレータその他の関係者にその発生をメール送信等の手段で連絡するアクションを行う．障害に対する具体的な対処は連絡を受けたオペレータ等の関係者に委ねられる．

調査を行ったデータセンタでは，監視対象となる機器が約 1 万台，発行されるメッセージ数は 1 時間あたり約 5,000 メッセージ，分析対象となる分析パターンは 7 パターンであり，データベースに登録されている分析ルールは約 1,000 個である．分析パターンのうち登録されている分析ルールの数が多い上位 3 つと分析ルール数の割合を表 3-1 に示す．

3.2.2 現状のメッセージ分析方式の課題

現状のメッセージ分析方式については，下記の 3 点の課題がある．

（1）分析パターン追加の課題

前述のように，現状のメッセージ分析ツールでは，分析パターンごとに分析するソフトウェアロジックを作り込む必要がある．そのため，新しい分析パターンの追加や分析パターンの変更が必要な場合には，ツールのプログラム修正，テストが必要となり，時間と工数がかかる．

（2）メッセージ分析ルール作成の課題

現状の方式では，過去の障害発生条件の解析に基づいて分析すべきメッセージの種別，

メッセージ発行元の情報などの情報を分析パターンのパラメタである分析ルールとしてデータベースに予め登録しておく必要がある．すなわち障害事例を蓄積することでブラックリストを充実させるアプローチとなっている．障害発生の未然防止の観点からは，当初は障害の可能性のある条件を分析ルールとして全て登録しておき，障害でないことが確認された分析ルールを順次削除することで分析精度を向上させるというやり方が考えられる．しかし，対象となるメッセージの種類，メッセージ発行元の情報が膨大であり，システム構成変更の度に障害の可能性のある条件の情報のメンテナンスを行うことは大きなコストがかかるため，現状では行われていない．

逆に障害ではない条件を追加していくホワイトリスト的アプローチも考えられるが，新たな分析パターンの追加となり，メッセージ分析ツールの改造が必要となるため，実施されていない．

(3) ルールメンテナンス運用上の課題

メッセージ分析ツールの目的は, 顧客に SLA で約束した時間内に障害発生を連絡することであり，リアルタイム性が要求される．そのため，メッセージ分析ツールは，全ての処理をインメモリで行うために，どのような条件でメッセージが発生したときに注意喚起アクションを実行するかというメッセージの分析ルールをツールの起動時に全てメモリ上に読込む．従って，分析ルールを変更する場合には，ツールの再立ち上げが必要となる．分析ルールの変更は，障害発生検知のための新たなルールの追加や，検知不要なルールの削除，アプリケーションプログラム，ミドルウェア，オペレーティングシステム等のバージョンアップによるメッセージの追加あるいは変更，あるいはハードウェア増設，交換といったシステム構成変更によるメッセージ発行元情報の変更を契機に行う必要がある．通常は，変更の契機ごとに分析ルールの変更を行うのではなく，複数の変更要因の整合性を確認し, 定期的に必要な変更をまとめて行っている．現状のメッセージ分析ツールにおいて，約 1,000 個の分析ルールを読込む場合，再立ち上げのための停止時間は分オーダとなる．監視対象となる情報システムは 24 時間連続運転が必要なシステムが多く, 分析ツールの停止に合わせて停止することは不可能であり，分析ツール停止中に 100 程度のメッセージが発行される可能性がある．現状では，再立ち上げ中はメッセージをファイルに蓄積しておき，ツールの再立ち上げ終了後に分析を行うといったリアルタイム性を犠牲にする運用をせざるをえない．さらに，分析ルールには一定時間継続して分析を行う必要のあるルールがあり，分析途中のタイミングで分析ツールの再立ち上げを行う場合には，再立ち上げ前に状態を保存しておき，再立ち上げ後に状態を引継ぐ処理が分析ツールで必要となる．

このような問題点を解決する手段として，刻々と発生するデータをリアルタイムに分析

```
register stream Message
(msgID INT,                    //  message ID
msgTIME TIMESTAMP,             //  time of message issuance
srcID INT,                     //  ID of message source
msgBody VARCHR(100));          //  message body
```

図 3-2 入力ストリームフォーマット

するのに適した技術として近年注目されているストリーム DB 技術を適用する方式を 3.3 節で提案する．

3.3 ストリーム DB を適用したメッセージ分析方式

3.3.1 ストリーム DB の概要

ストリームは，発生順に追加される時系列データ系列として定義され，仮想的に無限の長さを持つ．メッセージ分析の対象となるメッセージ情報のストリームは，図 3-2 に示すような CQL で定義される．図 3-3 に RDB とストリーム DB の比較を示す．RDB に対する問合わせは，蓄積されたデータに対して SQL を発行することで行われる．これに対して，ストリーム DB では理論上無限の長さを持つ時系列データ系列であるストリームを対象としている．ストリームに対して，データ系列の個数，あるいは発生時刻の範囲条件を与え

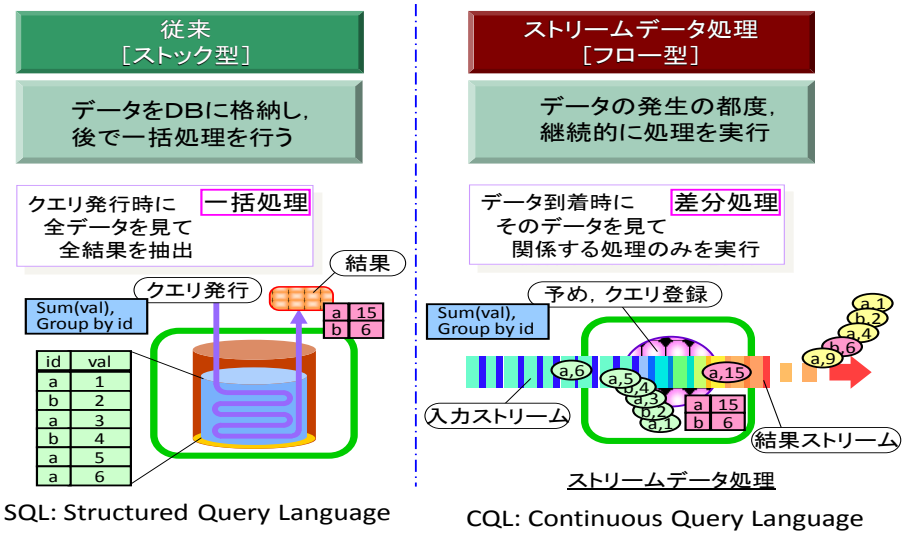


図 3-3 RDB と比較したストリーム DB の概要

ることで，有限個のデータ系列を抽出するスライディングウィンドウを定義する．スライディングウィンドウによって抽出された有限個数のデータ系列は，ストリーム定義によって与えられたスキーマを持つテーブルのリレーションとみなすことができる．ストリームDB では，この有限データ系列に対して RDB における SQL と類似の構文とセマンティクスを持つ問合わせ言語 CQL によって問合わせを行う．RDB では，SQL による問合わせの結果は，リレーションであるが，ストリーム DB の場合は，CQL による問合わせの結果はストリームまたはリレーションとなる．RDB では格納されているデータを更新したり削除する操作があるが，ストリーム DB では複数のストリームをマージして新たなストリームを出力ストリームとして生成することはできるが，入力ストリーム自身を更新する操作はない．ストリーム DB の処理系をストリーム DB エンジンと呼ぶ．ストリーム DB は，時系列データをリアルタイムに処理することを特色としており，性能確保のためにストリーム DB エンジンの処理は全てインメモリで処理されるのが普通である．

ストリーム DB 技術は，2000 年頃から米国を中心に研究が始まり，最近ではリアルタイム性確保のための QoS（Quality of Service）に関する研究[60]も行われている．日本においても活発に研究が進められている[61] [62][63][64]．商用の製品のストリーム DB エンジンとして，（株）日立製作所が uCosminexsus Stream Data Platform[65]を，Oracle 社が Oracle Fusion Middleware の CEP コンポーネント[66]の一部として販売している．

3.3.2 ストリーム DB を適用したメッセージ分析システムの概要

ストリーム DB をメッセージ分析に適用することで，ソフトウェアロジックの作りこみで実現していた分析ロジックを CQL 記述で実現でき，分析パターンの追加・変更が容易になることが期待できる．また，CQL 記述によって分析パターンを柔軟に追加でき，ブラックリストアプローチの分析パターンだけでなくホワイトリストアプローチの分析パターンの追加も従来方式と比較して容易であるため 3.2.2 節（2）で述べたメッセージ分析ルール作成の課題も解決できる．さらに，運用中に CQL を追加・変更可能なストリーム DB の実装が存在し，それを用いることで現状のルールメンテナンス運用上の問題を回避することができる．

ストリーム DB を適用したメッセージ分析システムの概要を図 3-4 に示す．図 3-1 のメッセージ分析ツールと分析ルールの部分が図 3-4 で置き換えられることになる．分析システムは，分析を実行するための CQL 及び分析に必要な付加情報を生成する CQL 生成ツールと，CQL 生成ツールによって生成された CQL と付加情報を用いてメッセージ分析を行う部分の 2 つに分けられる．

CQL 生成ツールは表 3-1 に示した分析パターンの種別及び分析パターン種別ごとに必要となるパラメタ情報を入力とし, 予め用意されている CQL のテンプレートにパラメタ情報を埋め込んで分析を実行するための CQL 及び付加的な情報を生成する.

生成する CQL 及び付加的な情報と生成ツールについては，3.4 節で詳細に説明する.

CQL を用いて実際にメッセージ分析を行う部分は，データ送信アプリケーション，ストリーム DB エンジン，結果利用アプリケーションの 3 つの部分から構成される．ストリーム DB エンジンは，CQL を実行する汎用のストリーム DB である．データ送信アプリケーションは，図 3-1 のメッセージ監視ツールが発行するイベントを入力してストリーム DB の入力となるデータストリームを出力する. 結果利用アプリケーションは, ストリーム DB によって障害発生のメッセージ発行パターンであると分析された結果の出力ストリームを入力して，図 3-1 の注意喚起ツールを起動する．データ送信アプリケーションと結果利用アプリケーションは，ストリーム DB エンジンが提供するインタフェースに合わせて作成する.

3.3.3 メッセージ分析へのストリーム DB 適用の課題

3.3.1 節で述べたように，ストリーム DB エンジンは，処理性能確保のため全てのデータをメモリ上で処理する構造となっている．3.2 節で述べたように，現状のメッセージ分析ツ

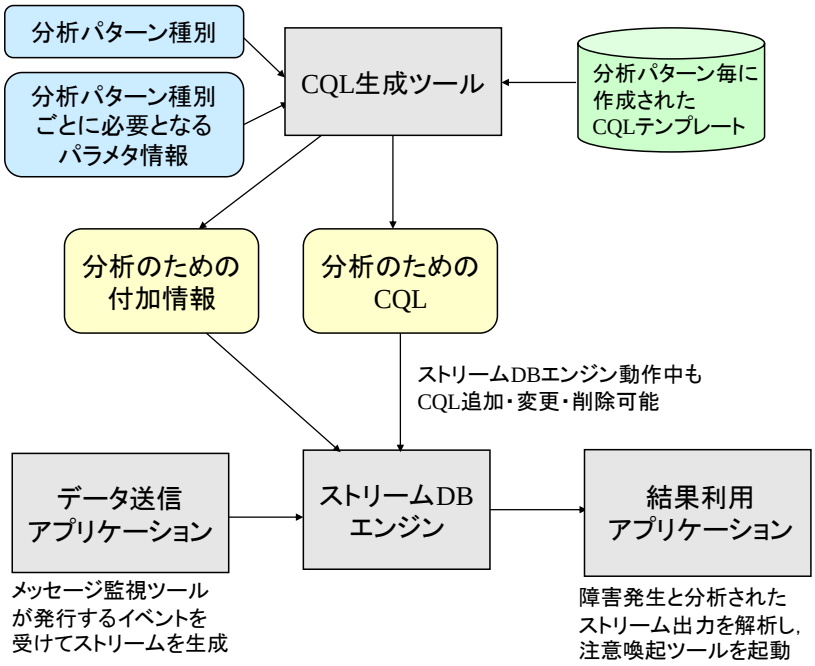


図 3-4 ストリーム DB を適用したメッセージ分析システム概要

ールも同様の理由でメモリ上で処理を行っている．ストリーム DB では処理するデータ量に依存して必要となるメモリとデータ量に依存せず固定的に必要となるメモリがあるが，固定的に必要となるメモリ使用量は，登録されているクエリの数，すなわち CQL の数にほぼ比例する．例えば，3.5 節の実証実験で使用したストリーム DB エンジンでは，CQL を１つ登録すると約 2MB のメモリを消費する．3.2.1 節でヒアリング調査した大規模データセンタでメッセージ分析ツールに登録されている分析ルールは約 1,000 個ある．分析ルールが複雑な場合には，1 つの分析ルールに相当する処理を行うために 10 個程度の CQL を登録する必要があるが，仮に分析ルールと CQL が 1 対 1 に対応するとした場合でも，約 2GB のメモリを消費する．今後，データセンタの大規模化が益々進展し，メッセージ分析ルールの数も増加しさらに複雑化することが想定される．そのような場合対応できるようにメッセージ分析システムのスケーラビリティを確保するためには，運用対象システムの規模の拡大に伴って分析ルールが増加しても CQL の数が増えないような CQL 設計上の工夫をしておく必要がある．

CQL は汎用的な問合わせ言語であり，その記述方法は自由度が大きい．メモリ使用量を意識した記述を徹底するためには CQL を自由に記述させない工夫が必要である．また，分析ルールは一度作成したら終わりではなく，新たな分析対象システムの追加や分析ルールの見直しなどのメンテナンス作業が必ず発生する．データセンタの運用管理者は必ずしも CQL に熟練しているとは限らないため，分析ルールの作成及びメンテナンスを効率的に行える CQL の自動生成ツールが必要となる．

3.4 メモリ使用量を意識した CQL の設計方針と自動生成方式

3.4.1 メモリ使用量を意識した CQL 設計方針

3.3.3 節で述べた課題に対応するための CQL の設計方針について，CQL の例を示し説明する．

まず，表 3- 1 の項番 1 の特定のメッセージが発行されたことを分析するパターンを考える．単純に考えるとこのパターンは，図 3-5 に示す CQL で分析するのが自然である．しかし，この方法では CQL 中に分析対象メッセージの情報を記述しているため，分析対象メッセージ数が増加するとそれに比例して必要な CQL が増加する．表 3-1 に示したようにこの

```
Let X be specifed message type ID,

register query Q1 istream (SELECT *
    FROM Message[ROWS 1] AS M WHERE M.msgID = X);
```

図 3-5 特定のメッセージ種別のメッセージ発行を検知するための単純な CQL

```
Definition of stream which holds list of message ID's to be analyzed.

register stream MessageList
(msgID INT );

CQL to check issued messages against list above.

register query Q1 istream (SELECT
    M1.msgID AS msgID,   M1.msgTIME AS mstTIME,
    M1.srcID AS srcID, M1.msgBODY AS msgBODY
  FROM Message[ROWS 1] AS M1,
        MessageList[NOW] AS M2
  WHERE M1.msgID = M2.msgID );
```

図 3-6 特定のメッセージ種別のメッセージを検知するための改良された CQL

分析パターンの分析ルールは全体の 70%を占めており，管理対象システムの拡大に伴いこの分析パターンの分析ルールの数も増大すると想定される．したがって，このような CQL を用いることは，メッセージ分析システムのスケーラビリティ確保上問題となる．

この状況を解決する手段を以下に述べる．図 3-6 に示すように，分析対象となるメッセージのメッセージ ID の情報を保持するストリームを定義する．評価で利用するストリーム DB エンジンは，ファイルから読込んだデータをストリームとして扱う機能を有している．この機能を利用して分析対象メッセージのリストをファイルに保存しておき，ストリームとして読込む．このストリームと管理対象システムが発行したメッセージのストリームの内容を突き合わせ，分析対象のメッセージの発行を分析する．この方式では，分析対象メッセージの数が増加しても分析に必要な CQL の数を一定に保つことができ，メッセージ分析システムのスケーラビリティを確保することができる．この分析対象メッセージのメッセージ ID のリストは，図 3-2 の CQL 生成ツールが CQL 以外に生成する「分析のための付加情報」の 1 つの例である．

図 3-5 の方式と図 3-6 の方式のメモリ使用量を 3.2.1 節で述べた大規模データセンタでの例で試算する．適用実験に使用したストリーム DB エンジンでは CQL 1 つ当たり約 2MB のメモリを使用する．図 3-5 の方式では約 700 個の分析ルール毎に別の CQL が必要となるため，メモリ使用量は約 1.4GB である．図 3-6 の方式では 1 つの CQL で処理できるためメモリ使用量は 2MB となる．従って，図 3-5 の方式と図 3-6 の方式では，メモリ使用量は約 700 倍の差となる．

図 3-5 の方式と図 3-6 の方式の計算量を比較する．分析対象メッセージ ID の数を n（3.2.1 節で述べた大規模データセンタでの例では，n=700）とする．図 3-5 の方式では計算量は n に比例する．図 3-6 の方式は，図 3-4 の方式と比較すると join 演算を含む複雑な検索条件となるが，ストリーム DB エンジンが nested-loops join アルゴリズム[67]を適用し，join 対象カラムがインデックス付けされる場合には計算量は log n のオーダとなる．計算量の比較においても，図 3-6 の方式がスケーラビリティの観点で優れている．

表 3-1 の項番 2 の分析パターンについても，単純に考えると CQL 中に分析対象メッセージ ID を指定するのが自然であるが，スケーラビリティの点では項番 1 の分析パターン同様問題となる．項番 1 の場合と同様に図 3-7 に示すようなメッセージ ID を明示的に指定しない CQL を使用するよう工夫することで，CQL の数の増加を抑止することができる．

図 3-7 の Q2 のクエリは，X 分間に出力されたメッセージをメッセージ ID とメッセージ発行元 ID で分類し，分類ごとの出現回数をカウントしその値を付加してストリームとして出力する．Q3 のクエリでは，出現回数が Y 回を越えているものだけを選択し，ストリームとして出力する．メッセージ種別とメッセージ発行元を指定する CQL ではメッセージ

```
Let X be specifed interval (in minute) and Y be specified threshold
of number of issuance of the message,

register  query Q2 istream
          (SELECT M.msgID AS msgID, M.srcID AS srcID,
           COUNT(*) AS cnt
           FROM Message[RANGE X MINUTE] AS M
           GROUP BY M.msgID, M.srcID );

register  query Q3 istream
           (SELECT msgID, srcID
             FROM Q1[NOW]
             WHERE cnt >= Y);
```

図 3-7 同一ホストから一定時間内に定められた回数を越えるメッセージ発行を検知するための CQL

種別数とメッセージ発行元数の積に比例するオーダの CQL が必要となる. 図 3-7 に示した CQL では時間と回数の条件が同一であればメッセージ種別とメッセージ発行元の組合せが異なっていても同一の CQL で分析が可能である.

ここで表 3-1 に示されていない分析パターンを実現するケースを考えてみる．図 3-7 に示す CQL は，条件を満たすメッセージを全て分析するが，特定のメッセージ ID を持つメッセージだけを分析したい場合が考えられる. CQL の処理結果は, ストリームになるので, 図 3-7 の CQL の出力ストリームに対して，図 3-6 に示した CQL と同様の CQL を適用することで実現できる. この場合も CQL の数は分析対象となるメッセージ種別の数に依存せず一定である.

さらに，別の分析パターンを考えてみる．図 3-6 に示した CQL は，分析対象とするメッセージのリストを用意しておくブラックリスト方式であるが，3.2.2 節（2）で課題として挙げたように，逆に分析対象から除外するメッセージのリストを用意するホワイトリスト方式を採りたい場合がある．この場合には，図 3-7 の CQL の出力ストリームに対して，図 3-8 に示すような CQL を適用することで実現できる. 図 3-8 の CQL を元にしてストリーム名称，フィールド名称を適切に変更することで，どのような場合のホワイトリスト方式にも適用可能である．後述の 3.5 節の適用実験結果で示すように CQL の追加は，プログラムロジックの追加・変更と比較して小さいコストで実装できる．そのため，図 3-8 の方法が「3.2.2 節（2）メッセージ分析ルール作成の課題」で挙げた課題の解決策となる.

表 3-1 の項番 3 の分析パターンは，図 3-9 に示す CQL で分析を行うことができる．この CQL では一定時間内に出現することを分析するメッセージ種別の情報は図 3-6 の CQL と同様に，ファイルに保存されている情報からストリームとして取り込む．この場合には，分析対象のメッセージ種別の組合せごとに別のストリームを定義する必要があるため，分分析対象のメッセージ種別の組合せごとに CQL を用意する必要がある.

表 3-1 の項番 4 のその他には，例えば複数のメッセージ種別が定められた順番で発行されているかどうかを分析するといった分析パターンが含まれる．また，現状のメッセージ分析ツールでは分析していないが，あるメッセージ種別のメッセージ発行後，別な特定のメッセージ種別のメッセージが発行されていることを分析するといった分析パターンも考えられる．本論文では CQL 例を示さないが，このような分析パターンについても CQL で記述可能である.

以上の検討から, メモリ所要量を削減するために必要な CQL の数をなるべく減らすための設計方針として,

Definition of stream which holds list of message ID's to be excluded from analysis,

```
register stream MessageWhiteList
(msgID INT );
```

CQL to output stream when the issued message does not appear in the list above,

```
register query Q1 istream (SELECT
      M1.msgID AS msgID,   M1.msgTIME AS msgTIME,
      M1.srcID AS srcID, M1.msgBODY AS msgBODY
  FROM Message[ROWS 1] AS M1,
       MessageWhiteList[NOW] AS M2
  WHERE M1.msgID = M2.msgID
  union
  Message[ROWS 1] );

Register query Q2 istream (SELECT
    msgID, msgTIME, srcID, msgBODY, COUNT(*) as cnt
    FROM Q1[NOW]
    GROUP BY msgID, msgTIME, srcID, msgBODY);

Register query Q3 istrem(SELECT
    MsgID, msgTIME, srcID, msgBODY
    FROM Q2[NOW]
    WHERE cnt = 1 );
```

図 3-8 与えられたリストに存在しないメッセージの発行を検知するための CQL

（設計方針 1）「メッセージ ID，メッセージ発行元 ID のように分析ルール毎に異なる情報はCQLに指定せず, 分析のための付加情報としてファイルに格納しCQLの共通化を図る.」が導かれる．図 3-6，図 3-7，　図 3-8，図 3-9 に示した CQL はいずれもこの設計方針に従っている．

また，図 3-6, 図 3-8 に挙げた CQL の検討から，新しい分析パターンの処理を設計する際の設計方針として，

（設計方針 2）「CQL の出力が再びストリームとなることに着目し，分析パターンの条件を分割して CQL を設計し，それを組合わせることで複雑な条件を実現する．入力ストリームが同じ CQL が複数回出現した場合には CQL を共有する」

が導かれる．（設計方針 2）はメモリ使用量削減のためだけでなく，新しい分析パターンを既存の分析パターンの組合わせで実現するためにも有効である．

Definition of stream which holds list of message ID's to be anlyzed whether all the messages in the list are issued or not.

```
register stream ExpectedMessageList
(msgID INT );
```

CQL to output stream when all the messages in the above list are issued within specified interval. Let N be the size of the list above and X be the specifed interval(in minute).

```
register query Q1 istream (SELECT
      M1.msgID AS msgID,   M1.msgTIME AS msgTIME,
      M1.srcID AS srcID, M1.msgBODY AS msgBODY
  FROM Message[RANGE X MINUTE] AS M1,
       ExpectedMessageList[NOW] AS M2
  WHERE M1.msgID = M2.msgID
  );

Register query Q2 istream (SELECT
   msgID, COUNT(*) as cnt
   FROM Q1[NOW]
   GROUP BY msgID);

Register query Q3 istrem(SELECT
   MsgID
   FROM Q2[NOW]
   WHERE cnt = N );
```

図 3-9 指定された時間内に一定回数発行されたメッセージを検知する CQL の例

図 3-10 に（設計方針 2）の適用例を示す．CQL と CQL の出力ストリームは 1 対 1 の関係なので，図 3-10 では両方を同じ図で表現している．メッセージをある条件でフィルタリングし，その結果をブラックリストでさらにフィルタリングする分析パターンと，ホワイトリストでフィルタリングする分析パターンが必要な場合, 図 3-10 に示すような処理で実現できる．この場合，最初のフィルタリングの CQL は共用できる．また，ブラックリストまたはホワイトリストによるフィルタリングは CQL あるいは CQL テンプレートを設計する際に部品的に使うことができる．

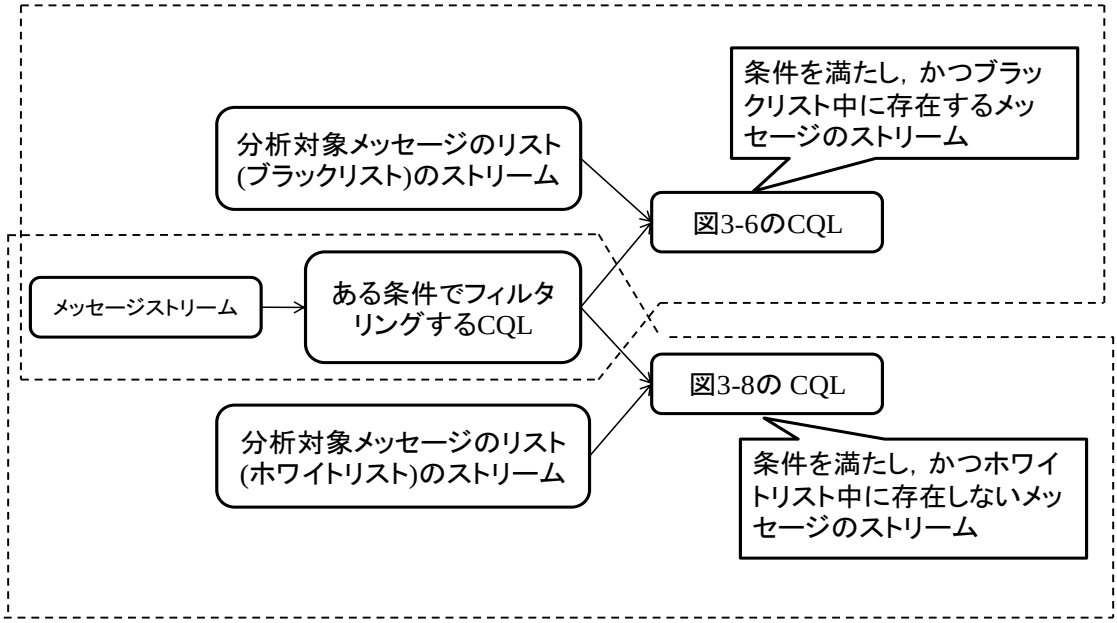


図 3-10 設計指針 2 の適用例

3.4.2 設計方針の評価

3.2.1 節で説明した大規模データセンタでは，分析ルールの数は約 1,000 であり，単純に個々の分析ルールを CQL で実現する場合には，CQL を登録するだけで約 2GB のメモリを消費する．本章で提案する CQL の設計指針 1 に従って CQL を作成する場合，全体の 70%を占める表 1 の項番 1 の分析ルールは単独の CQL で実現される．全体の 25%を占める表 1 の項番 2 の分析ルールは，時間及び回数の条件が同一であれば同じ CQL で実現される．残りの 5%は，分析ルールと CQL は 1 対 1 である．表 1 の項番 2 で同じ時間及び回数の条件を持つ分析ルールは平均 10 個程度であるため，本節で提案する CQL 設計方針に基いて作成した CQL を登録することによるメモリ使用量は，

$2 + 2\times(250\div10) + 50\times2 = 152$（MB）

となり，単純な CQL 発行と比較し約 1/13 と，メモリ使用量を大きく低減できる．単純生成方式と提案方式のメモリ消費量の比較を図 3-11 に示す.

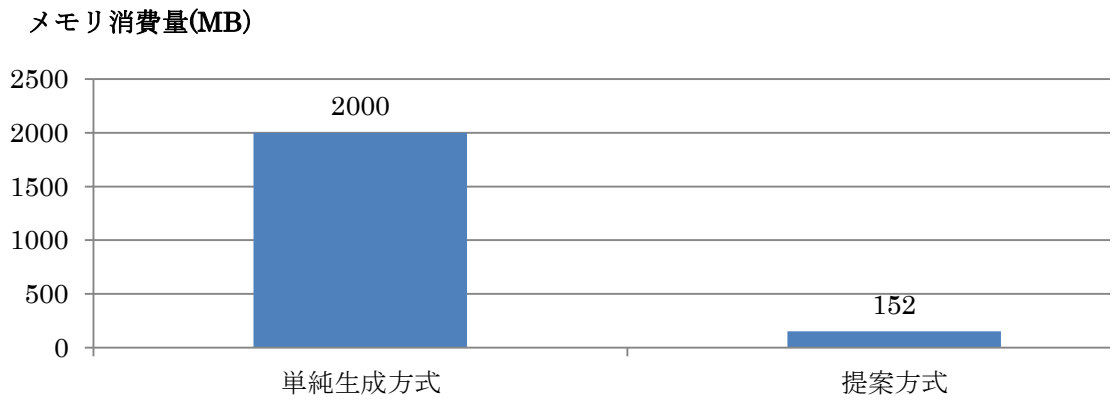


図 3-11 単純生成方式と提案方式のメモリ消費量比較

3.4.3 CQL テンプレートと自動生成方式

（設計方針 1）に従って CQL を生成する場合，図 3-6，図 3-7 の CQL 例からわかるように，分析ルールと生成される CQL の関係は 1 対 1 とならず，多対 1 の関係になる．生成対象の構造をテンプレートとして与え，可変部分をパラメタとして与えることで自動生成を行う手法とツールはさまざま提案され実装されている．生成対象（この場合は生成される CQL）単位に，生成に必要なパラメタ（この場合には分析ルールの情報）を与えるという方式が一般的である. この方式では, 分析ルールとして管理している情報を, 生成対象 CQL 単位に整理し直して自動生成ツールに生成パラメタとして与える必要がある．分析ルールをメンテナンスするデータセンタ運用管理者の観点からは，管理の対象は各々の分析ルールであり，分析を実行するための CQL ではないので，このような情報の再整理は CQL 自動生成ツール側で行うべきである．

これを実現するための CQL 自動生成方式を図 3-12 に示す．入力となる情報は，分析ルール単位で管理されており，分析ルール ID，分析パターン ID，対象メッセージ ID，対象発行元 ID，時間条件，回数条件その他のデータから成る．3.3.2 節で説明したように分析ルールがどのような情報を持つかは，分析パターンによって異なる．生成処理は，分析パターン毎に，設計方針 1 及び設計方針 2 に従って設計されたテンプレートに分析ルールの情報を適用して実行する．例えば，表 3- 1 の項番 1 の分析パターンの場合には，図 3-6 の CQL を生成するので，この分析パターンの全ての分析ルールの対象メッセージ ID の情報は付加情報として生成され， CQL はテンプレートで用意したものがそのまま生成される．表 3-1 の項番 2 の分析パターンの場合には，この分析パターンの分析ルールは時間条件及び回数条件で分類され，分類された単位毎に付加情報と CQL が生成される．この場合のテンプレートは，図 3-7 に示した CQL の X 及び Y を分析ルールの情報で置換する対象とし

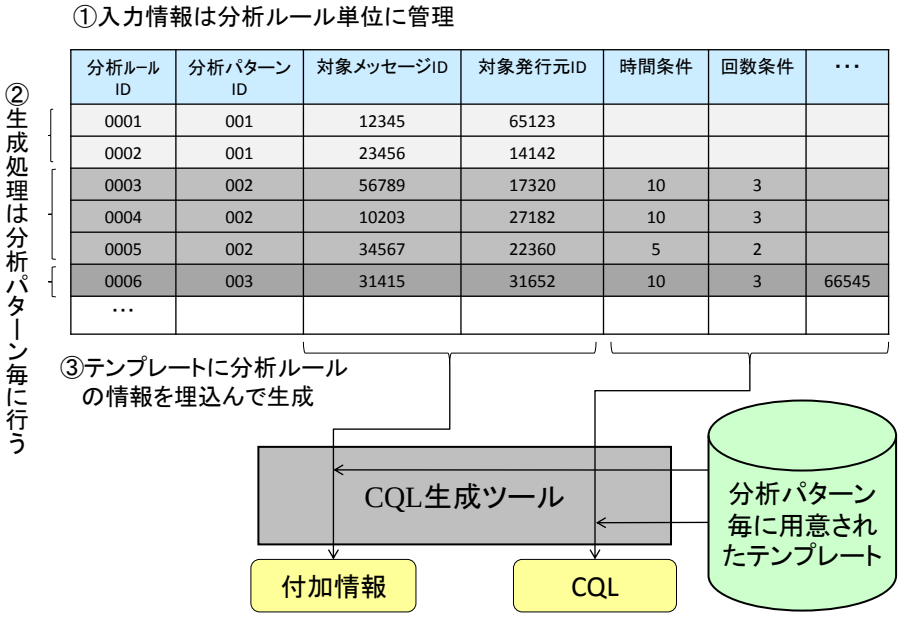


図 3-12 設計方針に基く CQL 生成方式

て表現したものとなる．表 3-1 の項番 3 の分析パターンの場合には，図 3-9 に示した CQL の最終行の N とストリーム名称である ExpectedMessageList の部分がテンプレート中の置換する対象となる．図 3-9 の N の部分はそのストリーム名称で読み込む付加情報として生成したデータの個数で，ExpectedMessageList の部分は，CQL 生成ツールが付与したストリーム名称で置き換えられる．この場合には，分析ルール毎に CQL と付加情報が生成される．

図 3-10 に示したような例の場合には，点線で囲んだ単位でテンプレートを作成して登録することにより CQL の共用を実現する．この場合，分析パターンは，共通化される可能性のある抽出条件とホワイトリストで除外する条件からなる分析パターンと，共通化される可能性のある抽出条件とブラックリストで選択する条件からなる分析パターンの 2 つの分析パターンが用意される．CQL 生成は，分析パターン毎に行うので，共通化される可能性のある抽出条件を持つ分析パターンについては，一方の分析パターンの CQL 生成時に抽出条件の情報を記録しておき，もう一方の分析パターンの CQL 生成時に，同一の抽出条件の CQL が生成済みであれば，生成済みの CQL の出力ストリームを入力としてホワイトリストまたはブラックリストとの照合を行う CQL を生成することで，共通化される CQL を重複して生成しないように制御する．

この自動生成方式では，分析ルール単位に登録されている情報のリストから，分析パタ

ーン毎に必要な情報を抽出して生成処理を行う．そのため，分析ルールの登録順序によって生成される CQL が影響を受けることはなく, 分析ルールの保守は分析ルール毎に独立して行うことができる.

3.5 ストリーム DB によるメッセージ分析方式の適用実験

3.5.1 適用実験環境と適用分析パターン

3.2.1 節 でヒアリングを実施したデータセンタにおいて，ストリーム DB によるメッセージ分析方式の適用実験を行った．このデータセンタでは，図 3-1 に示したメッセージ分析システムが稼動中であるが，これと並行してストリーム DB を適用したメッセージ分析システムを稼動させて実験を行った.

実験環境は，CPU （Intel® Xeon® 5355 クワッドコア（2.6GHz）×2），メモリ 2GB の PC に VMWare® ESX™ 3.5 を用いた仮想環境上でゲスト OS として Windows® Server 2003 R2 Enterprise Edition を動作させ，ストリーム DB エンジンとして，（株）日立製作所製の uCosminexus Stream Data Platform 01-01 を使用している.

今回の実験の目的は，以下の２つである．まず，今回提案したストリーム DB を用いたメッセージ分析方式が現行方式の課題を解決できることを実際に運用中の環境で評価すること．さらに，評価実験で良好な結果が得られた場合には，実際の運用への適用を検討することである．そのため，現状のメッセージ分析ツールでサポートされていない分析パターンを今回提案するストリーム DB を用いた方式で開発し，3.2.2 節で述べた現行メッセージ分析方式の課題の（1）分析パターン追加の課題と（3）ルールメンテナンス運用上の課題について評価を行った.

評価のために開発した分析パターンは，現状のメッセージ分析ツールによる運用において課題となっている現象を解決するためのものである．この分析パターンに当てはまる状況が発生すると，現行のメッセージ分析システムでは，管理対象システム全体のメッセージ分析がスローダウンする．この場合に，メッセージ分析システムのスローダウンから速やかに回復するために，メッセージ多発発生時にその発行元を速やかに特定し，その発行元を監視対象からはずす必要がある．具体的には，分析対象メッセージ ID，発行元 ID を指定せずに，一定時間内に規定以上の数のメッセージが同一発行元から発行され，しかも一定時間以内にメッセージの発行数が規定の回数以下に復帰しないケースを分析するものである．この状況の発生と発行元の分析をストリーム DB を適用したメッセージ分析ツールを現行のメッセージ分析システムと並行して動作させて行う.

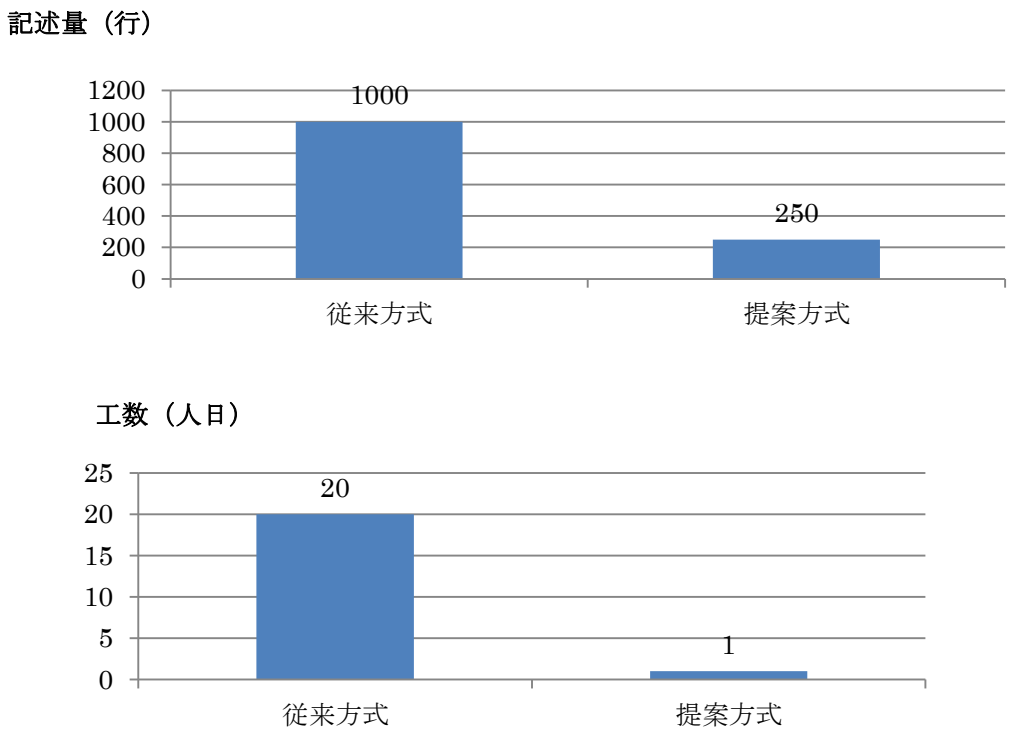


図 3-13 パターン追加のための記述量と工数の比較

3.5.2 分析パターン追加の容易性の評価

3.5.1 節で述べたように適用実験では新たな分析パターンを適用しているが， CQL 記述量は約 250 ステップであり，開発工数は約 1 人日である．既存の分析ツールに機能追加を行う場合の追加処理の見積りは，Java 言語で約 1,000 ステップであり，開発工数の見積りは約 1 人月であり，ストリーム DB を適用した場合の方が分析パターンの追加が容易であると評価できる．従来方式と提案方式のパターン追加のための記述量と工数の比較を図 3-13 に示す.

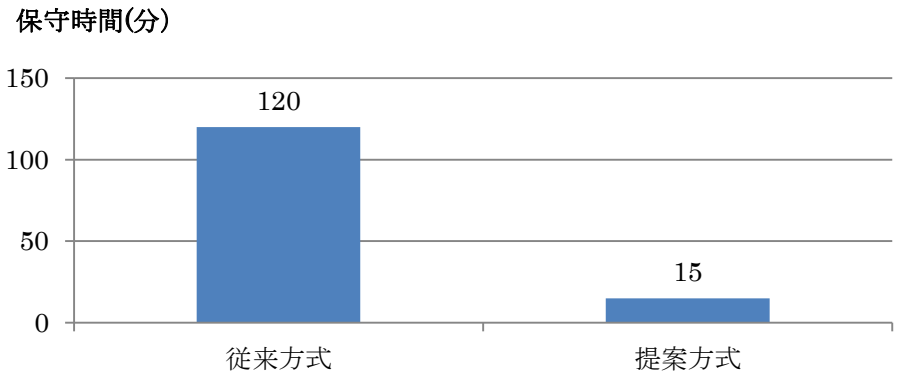


図 3-14　分析ルール保守時間の比較

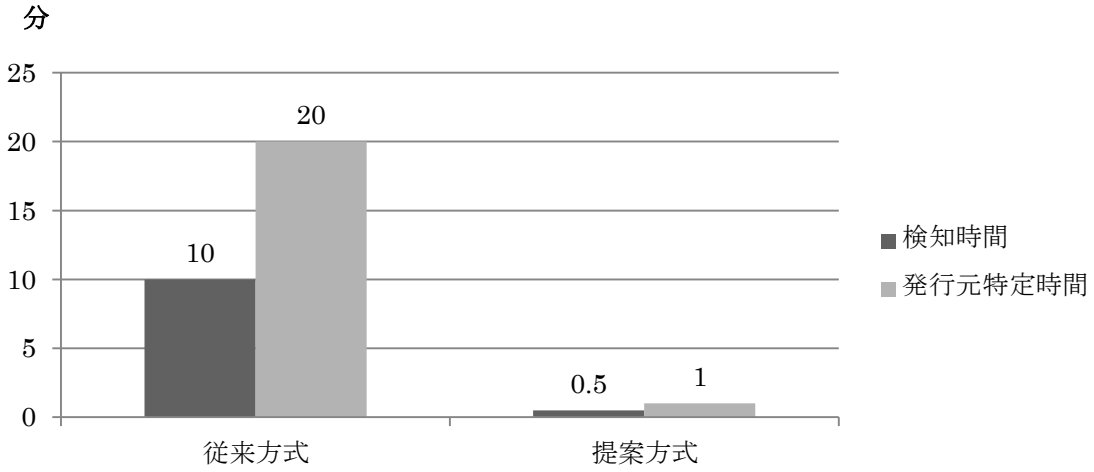


図 3-15 検知時間と発行元特定時間の比較

3.5.3 分析ルールのメンテナンスの容易性の評価

適用実験においては，時間条件，回数条件を運用管理者が調整できるように，CQL 自動生成ツールを提供している．現行のメッセージ分析ツールでは，分析ルールのメンテナンスに約 120 分掛かっているが，ストリーム DB を適用した実験環境の分析ルールのメンテナンスは約 15 分で行うことができる．本番環境と実験環境という差はあるが，ストリーム DB を適用するメッセージ分析システムでの分析ルールのメンテナンスの容易性を示している．従来方式と提案方式の保守時間の比較を図 3-14 に示す．

3.5.4 適用実験評価のまとめ

分析パターン追加の容易性，分析ルールのメンテンスの容易性に関して，ストリーム DB を適用したメッセージ分析方式と従来方式を比較し，前者の優位性を示した．特定発行元からのメッセージ多発の状況が発生すると，現行のメッセージ分析システムではメッセージ分析システム自体がスローダウンし，この状況の発生を検知するまでに約 10 分かかっていたが，ストリーム DB の適用により 30 秒で検知できるようになった．さらに，現行では検知の時間も含めて約 20 分かかる発行元の特定が 1 分以内でできている．以上より，適用実験の目的である迅速な対応に有効であることを確認した．従来方式と提案方式の比較を図 3-15 に示す．

適用実験の結果，現行のメッセージ分析システムの課題が解決されていることから，実適用することになった．現行システムと並行で動作させることが可能であるため，現行システムと並存させ緩やかに移行を図ることになっている．

3.6 結言

データセンタにおける運用管理でのメッセージ分析方式の現状の問題点の解決策としてストリーム DB を適用する方式を提案した．その際に，メッセージ分析ツールのスケーラビリティ確保の観点で課題となるストリーム DB のメモリ消費量を削減するための CQL 設計方針を提案し，分析ルール単位に管理する分析ルールの情報から設計方針に基く CQL を生成するための CQL 自動生成方式を提案した．提案方式を，実際に大規模データセンタの運用環境で適用実験を行い，提案方式の有効性を示した．

本章で提案した方式は，3.5 節で述べた適用実験を行った大規模データセンタにおいて，既存のメッセージ分析ツールを補完する形で導入され，実適用が始まっている．メッセージを分析して障害発生を検知することはほとんどの情報システムで有効であるため，システム運用管理製品の一部の機能として本章で提案した方式を提供する検討も行われている．

メッセージ分析のパターンを一般化して標準的な CQL テンプレートを提供することで，本方式をより簡単に効率的に適用できるようにすることが，普及を図るための課題である．

第 4 章
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

4.1 緒言

本章では，システム運用管理の高度化の中で，運用手順開始のきっかけを与える「イベント検知」の高度化，特に障害発生の予兆検知に関して論じる．

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． 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．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コンピューティングサービス[(68)]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．

リソース不足が発生した後，オペレーティングシステムや Web サーバ等のログを関連付けて解析することによって，キャパシティプラニング時の想定と実際のアプリケーションの動作が異なっていることがわかることがある．本章では，これらのログ情報をリアルタイムに解析することで，システムの状態を監視し，リソース不足の予兆を検知する方法を提案する．複数のログ情報を関連付けて解析する処理は，3.1 節で説明した CEP 技術の適用分野である．3.1 節で説明したのと同じ理由で，本章でも時系列データの解析を行うことができるストリーム DB をリソース不足の予兆検知に使用する．

本章の構成は，以下のとおりである．4.2 節で現状行われているキャパシティプラニングの方法とその問題点を示す．4.3 節では，ストリーム DB を適用した障害予兆検知方式を提案する．4.4 節で 4.3 節で提案した方式の有効性をプロトタイプによる実験で検証する．4.5 節では 4.3 節で提案した方式に統計的手法を適用する方法を提案し，その効果を実験により検証する．

4.2 キャパシティプラニングの方法と問題点

4.2.1 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．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 4- 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処理単位毎の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．キャパシティプラニング支援ツールを利用して実行環境の定義パラメタを生成する場合の状況を図 4-1 に示す．この場合，運用中の IT リソース不足の監視は，同時アクセスユーザ数，およびメモリ使用量を閾値監視することで行われ，キャパシティプラニングの根拠となっている設計基準値の監視は，複数のログ情報をマージして処理する必要があり実施されていない場合が多い．この例の場合には，同時アクセスユーザ数を Web サーバのログから取得し，業務アプリケーション全体のメモリ使用量を OS のログから取得して，タイムスタンプで時刻を合わせてユーザ数あたりのメモリ使用量を計算する必要がある．

表 4-1 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	―

図 4-1 従来のキャパシティプラニング

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 q 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，想定どおりに動作すれば，

$$R = F + qr \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (4\text{-}1)$$

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 Q とするとき安全係数を α （ >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．

$$R_{\max} = \alpha(F + Qr) \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (4\text{-}2)$$

4.2.2 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でも，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 Q を超えてしまう場合.

(2) 式（4-1）の r が過小評価の場合.

(3) 式（4-1）のように必要リソース量が q の 1 次式にならず 2 次以上のリソース量が必要となる場合.

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，アプリケーションプログラム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（4-1）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（4-1）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．次節以降で，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．

4.3 ストリーム DB 適用によるリアルタイム障害予兆検知

4.3.1 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 4-2 に本章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．4.2 節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．

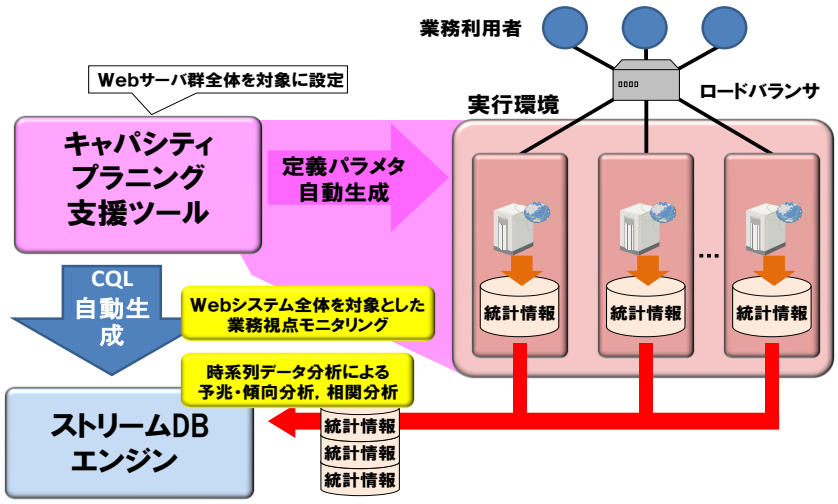


図 4-2 ストリーム DB によるリアルタイム予兆検知方式概要

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 エンジンの入力とし，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．

4.3.2 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．

4.2 節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．4.2 節で示した例で考えると，式(4-1)を変形して次の式が得られる．

$$r = \frac{(R-F)}{q} \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (4\text{-}3)$$

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 q 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 は式（4-4）となり，リアルタイムに CQL で計算できる値である．

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

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．実際の運用中に設計基準値以上にリソースを消費していないためには，次の不等式が成立している必要がある．

$$r \geq \frac{(R_m - F)}{q} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (4\text{-}5)$$

R_m , q はログ情報からリアルタイムに取得できるので，式（4-5）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内で動作しているかどうか判断することができる．式（4-5）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（4-5）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（4-5）が成立しているかどうかを判定するだけでなく，式（4-4）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（4-4）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（4-5）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，アプリケーションプログラム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.

4.3.3 CQL 自動生成

4.3.2 節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.

4.3.2 節の（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から CQL が処理対象とするスライディングウィンドウの定義が定まる．スライディングウィンドウは，理論的には無限長のデータ系列であるストリームの中から CQL による問合わせを行う対象となる有限のデータ系列を抽出するために用いられ，データ系列の個数あるいはデータ系列の発生時刻の条件によって定義される．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，4.3.4 節で説明する.

4.3.2 節の（2）の基準値相当値の監視についても，式（4-4）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.

4.3.2 節の（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.

4.3.4 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． 4.3.3 節で説明したように 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 4-3 の方式 1 参照.）

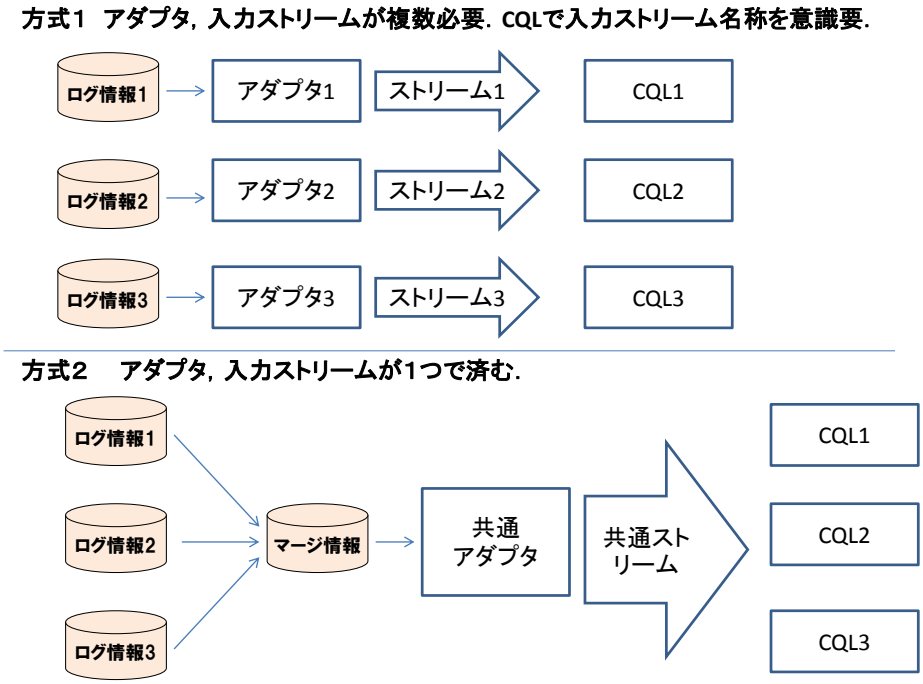


図 4-3 ログデータ取り込み方式の比較

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 4-3 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.

4.4 プロトタイプ実装による実験

4.3 節で提案した予兆検知方式のプロトタイプを構築し，予兆検知の実験を行った. 図 4-4 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 4.3.4 節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/アプリケーションサーバ単位のログ情報を作成した．さらに，3 台の Web/アプリケーション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リーム化し，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

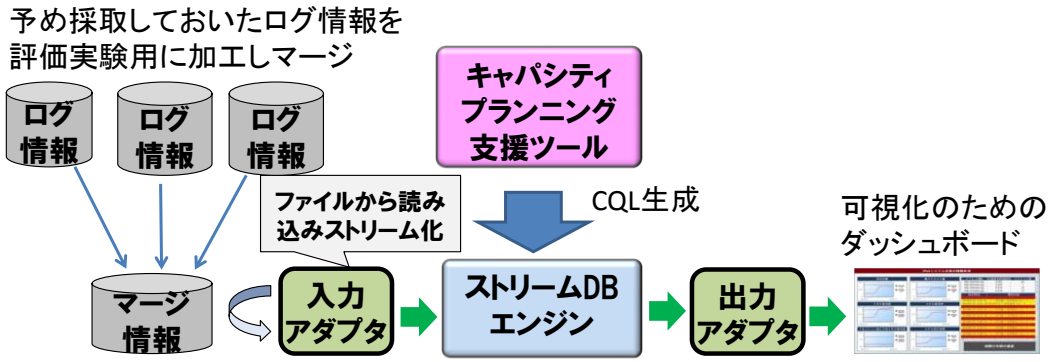


図 4-4 プロトタイプ実験環境の概要

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 4-5 に示す．図 4-5 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 4-5 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．

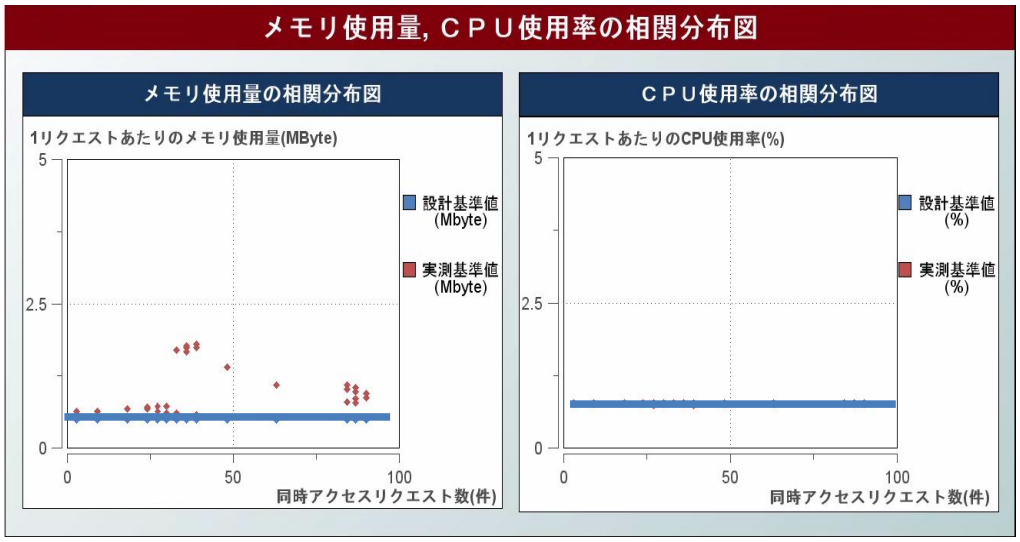
図 4-6 はシステムがキャパシティプラニングの想定通りに稼働している場合の例を示している．正常動作時には，実測基準値が設計基準値を下回っており，キャパシティプラニング時の想定に従って動作していることがわかる．

4.5 稼働情報のリアルタイム分析への統計的手法の導入

4.5.1 統計的手法導入の目的

4.3 節で提案した手法は，設計基準値と実測基準値の偏差を閾値監視あるいは線形外挿によって検出する手法である．シックスシグマ[69]等の製造業における品質管理手法においては，値のばらつきを統計的手法に基いて管理することが一般的となっている．本節では実測基準値の設計基準値からの偏差をリアルタイムに統計的解析を行い，キャパシティプラニングの想定とおりに稼働しているかどうかを確認することで，システムの状態を的確に把握し，安定的なシステム稼働を実現するための手法を提案する．

統計的手法を導入することで，検定の危険率によって偏差の検出の精度を定量的に制御することができる．危険率を大きくとることで，早期に小さな偏差を検出することができるが，実質的に問題とならない偏差も検出される，すなわち誤報の可能性も大きくなる．逆に危険率を小さくすることで，誤報の可能性は小さくなるが，偏差の見逃しあるいは検出の遅れの可能性が大きくなる．危険率を用いて，データセンタにおける IT リソース管理レベルをユーザに約束する SLA に沿って定量的に定めることができる．サービスレベルの高い契約の場合には，リソース不足の検知について危険率を大きく設定し小さな偏差に対しても早期に対処を行い，サービスレベルの低い契約の場合には危険率を小さく設定し，



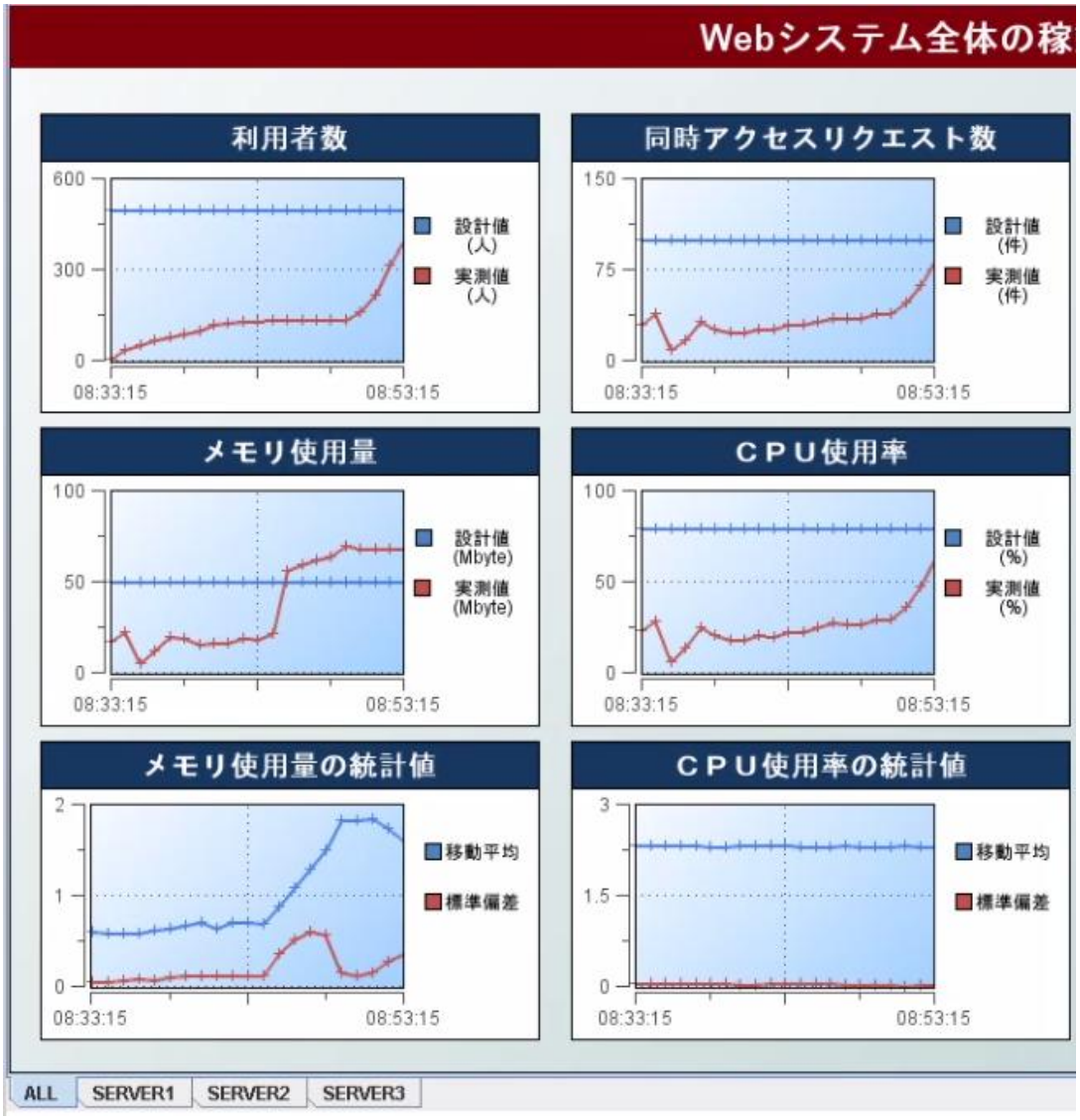


図 4-5 設計基準値と実測基準値の乖離による障害予兆検知例

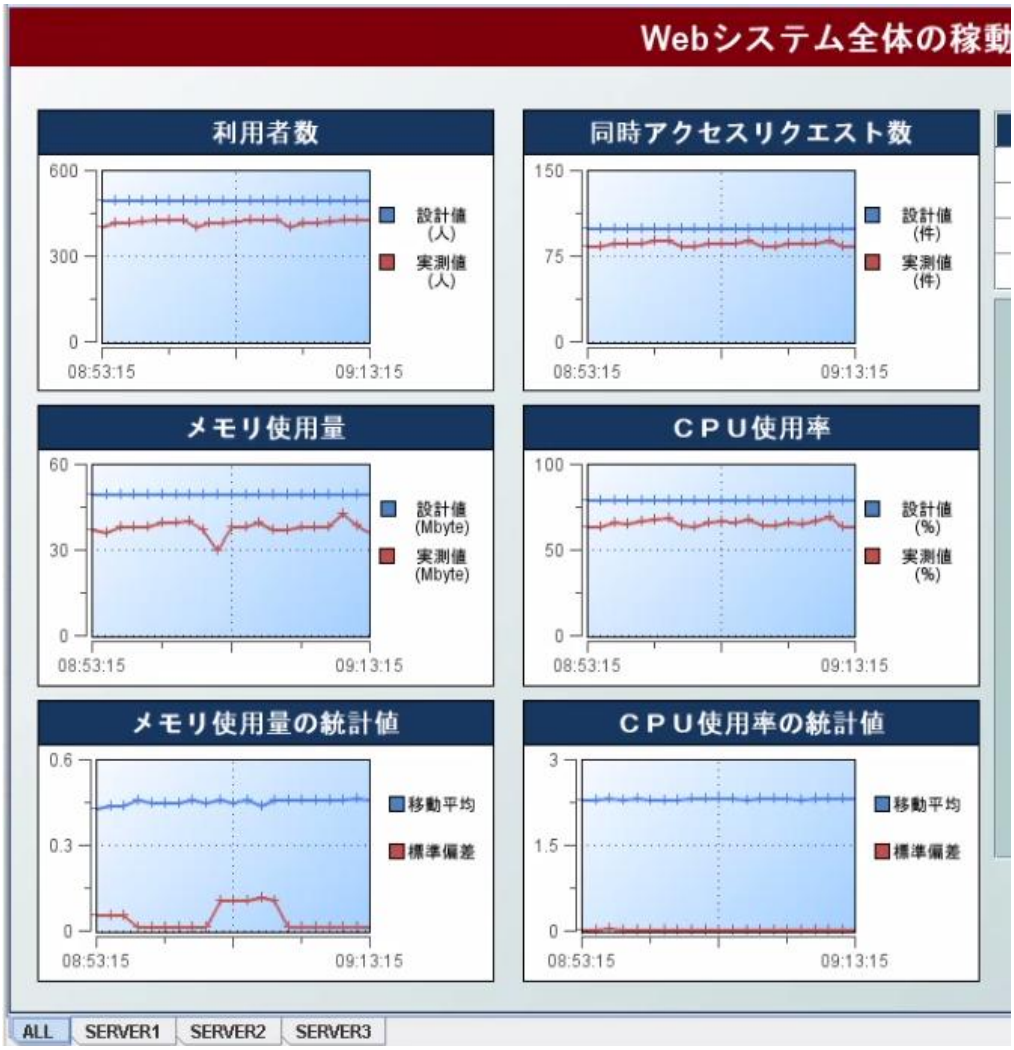


図 4-6 実測基準値が設計基準値以内となっている例

リソース不足発生の確信度が高くなってから対処を行うといった定量的な制御が可能となる．リソース過剰の検知については，逆にサービスレベルの高い契約では，危険率を小さく設定し，リソースが過剰であることの確信度が高くなってからリソース縮退を行い，サービスレベルの低い契約では危険率を大きく設定し早めにリソース縮退の処理を行うという制御が可能である．サービスの価格設定をサービスレベルに応じて設定する場合の定量的な根拠を危険率によって与えることができるようになる．

4.5.2 実測基準値が設計基準値と等しいことの検定

観測値が一定の確率分布に従うと仮定した場合に，実測された値を元に危険率 α で検定する方法が種々知られている．

本節では，設計基準値と実測基準値の偏差が検定の対象となる．実測基準値は，トランザクション当りの CPU 使用量，メモリ使用量などであるが，実際にはシステム全体の CPU 使用量，メモリ使用量をトランザクション数で割って平均を求める．Web アプリケーションにおいては，性能上の観点からトラクンザクション毎にオペレーティングシステムのプロセスを割り当てず，スレッドを割り当てて処理を行うことが一般的であり，ログ情報として取得するのは通常プロセス単位の情報だからである．ある時点での同時トランザクション数をpとするとき，各トランザクションのリソース使用量を確率変数X_i（$i = 1, \cdot\cdot, p$）で表す．各X_iは互いに独立であり，平均 m，分散 σ^2 の同じ分布に従うとしてよい．中心極限定理[70]によりpが十分大きいときX_iの平均 $\bar{X}$は，平均 m，分散 σ^2/p の正規分布で近似できる．リソース不足が問題となるようなアプリケーションプログラムでは，同時トランザクション数 p は数千，数万を超えると想定されるので，実測基準値が正規分布に従うとして検定を行ってよい．

キャパシティプラニングの想定とおりに稼働しているかどうかを判定するための手段として，一定時間内の実測基準値の分布の平均が設計基準値と等しいかどうかを検定する．ここで，一般的に実測基準値の測定時間間隔は，個々の Web アプリケーションのトランザクション処理時間よりも十分大きい．4.5.5 節の実験では測定時間間隔は 10 秒であり，Web アプリケーションのトランザクション処理時間は，通常 1 秒未満である．よって，実測基準値の間には相関はなく，それぞれ独立であるとしてよい．

実測基準値の平均が m に等しいかどうかを検定する．実測基準値の確率分布の分散は未知のため，危険率 α%の両側検定によって，実測基準値の平均値を$\bar{x}$，検定する対象の実測基準値の数を n とするとき，

$$m - t_{\frac{\alpha}{2}}(n-1)\frac{u}{\sqrt{n}} < \bar{x} < m + t_{\frac{\alpha}{2}}(n-1)\frac{u}{\sqrt{n}} \quad \cdots\cdots\cdots\cdots \quad (4\text{-}6)$$

が成り立つ時に実測基準値の分布の平均は m と等しいと検定される．

ただし，$t_{\frac{\alpha}{2}}(n-1)$は，自由度 n-1 の t 分布[76]の$\frac{\alpha}{2}$パーセント点，$u^2$は実測基準値を x_1，x_2，$\cdot\cdot$, x_n とするとき，

$$u^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

で与えられる不偏分散の値である．

リソース不足になる危険性だけを検定したい場合には，右片側検定を用いる．

この場合には，危険率 α%として，

$$\bar{x} < m + t_{\alpha}(n-1)\frac{u}{\sqrt{n}} \cdots\cdots (4\text{-}7)$$

ならば，実測基準値の分布の平均は設計基準値を上回らないといえる．

逆に，設計基準値が過大でないかどうかを検定したい場合には，左片側検定を用いる．

この場合には，危険率を α%として，

$$\bar{x} > m - t_{\alpha}(n-1)\frac{u}{\sqrt{n}} \cdots\cdots (4\text{-}8)$$

ならば，実測基準値の分布の平均は設計基準値を下回らないといえる．

n が十分大きいとき（実用上 30 以上の場合）には t 分布t(n)は，正規分布で非常に良く近似でき，不偏分散は観測された値の分散で近似できることが知られている[71]．

よって，検定に使用する実測基準値の数を十分大きくとることで，式（4-6）（4-7）（4-8）は，それぞれ式（4-9）（4-10）（4-11）で置換えることができる．ただし，z_{α} は正規分布の α パーセント値，s^2 は実測基準値の観測値の分散とする．

$$m - z_{\frac{\alpha}{2}}\frac{s}{\sqrt{n}} < \bar{x} < m + z_{\frac{\alpha}{2}}\frac{s}{\sqrt{n}} \cdots\cdots (4\text{-}9)$$

$$\bar{x} < m + z_{\alpha}\frac{s}{\sqrt{n}} \cdots\cdots (4\text{-}10)$$

$$\bar{x} > m - z_{\alpha}\frac{s}{\sqrt{n}} \cdots\cdots (4\text{-}11)$$

上式からわかるように，分散が大きくなると許容範囲が大きくなる．s^2 は実測基準値の観測値の分散としているので，　同時に分散の閾値監視を行うことで，許容範囲が拡大しすぎることを防止する．

4.5.3 危険率の設定

式（4-9）（4-10）（4-11）の計算を行うにあたり，危険率の設定が問題となる．なるべくリソース不足が発生しないように監視するためには，たとえ検定の精度が下がり，実際には実測基準値の分布の平均が設計基準値を上回っていない場合の誤検出があっても構わないので，危険率を大きく，すなわち信頼係数を小さく設定することになる．逆に，設計基準値が過大であることを検定する場合には，危険率を小さく，すなわち信頼係数を大きく設定することになる．

システムテストの段階で，危険率を大きめにとって検定を行い，その結果から設計基準値の修正を行い，実運用ではシステムテストの状況に応じて危険率を小さく設定して検定を行うのが現実的である．

4.5.4 検定のための CQL のウィンドウサイズ

4.5.2 節の議論の前提は，検定に使用する実測基準値の数 n が十分大きいことである．検

定のための CQL では，n は CQL が問合わせ適用の対象とするデータ系列の数を規定するウィンドウサイズで与えられる．ストリーム DB はリアルタイム性能を確保するために全ての処理がデータをメモリ上に展開するため，スケーラビリティ確保の観点でメモリ使用量を小さく抑えることが重要である. 4.5.5 節の実験で用いるストリーム DB エンジンでは，入力ストリームの 1 つのデータのサイズを s，ウィンドウサイズを W とするとき，1 監視対象システム当り 2s･W + 100KB のメモリをスライディングウィンドウで消費する．実験で用いた入力ストリームでは，s は 1.25KB であるので，1 監視対象システム当り 2.5W+ 100KB となる．実験で使用した CQL は 1 監視対象システム当り 7 個であり，CQL 登録に必要なメモリ量は 1 監視対象システム当り 14MB である．n が 100 未満であれば，スライディングウィンドウで消費するメモリ量は，CQL 登録で消費するメモリ量の 2.5%程度であり問題ないが，n=1,000 以上のウィンドウサイズが必要となると約 20%程度となりストリーム DB エンジンのメモリ消費量全体に与える影響が大きくなる．そのため，十分な検定精度を担保しメモリ消費量を低く抑えられるよう適切に設定することが重要である．

4.5.5 実験の想定環境とテストデータ

実験で想定する環境の構成は図 4-4 に示したものと同じである．ロードバランサで負荷時実行トランザクション数，メモリ消費量，CPU 使用率の情報がログ情報として定期的に出力されており, この情報をマージして入力ストリームとしてストリーム DB に取り込む. 実験では，マージ後の情報を想定したテストデータを作成し実験を行った．これらのデータの作成にあたっては，Web システムのデバッグおよびトラブルシューティングに関する文献[72]を参考にしている．

図 4-7 から図 4-9 に実験で使用したデータを示す．いずれも 10 秒に 1 回の間隔でログ情報を読み出しストリーム DB で分析する入力ストリームとしている．図 4-7 は，アプリケーションプログラムの不具合等によって 13 時にメモリリークが発生し, トランザクション当りのメモリ消費量が増大するケースを想定したデータである．ところどころ値が減少しているのは，ガーベージコレクションが発生して一時的にシステム全体のメモリ消費量が減少している状況を示している．メモリリーク発生までは設計基準値である 0.25MB/トランザクションを平均とする正規分布を想定している．

図 4-8 は，異常なスパイクを含むトランザクション当りの CPU 利用率を示している．正常時は，設計基準値であるトランザクション当り 0.1%の CPU 利用率を平均とする正規分布を想定している．

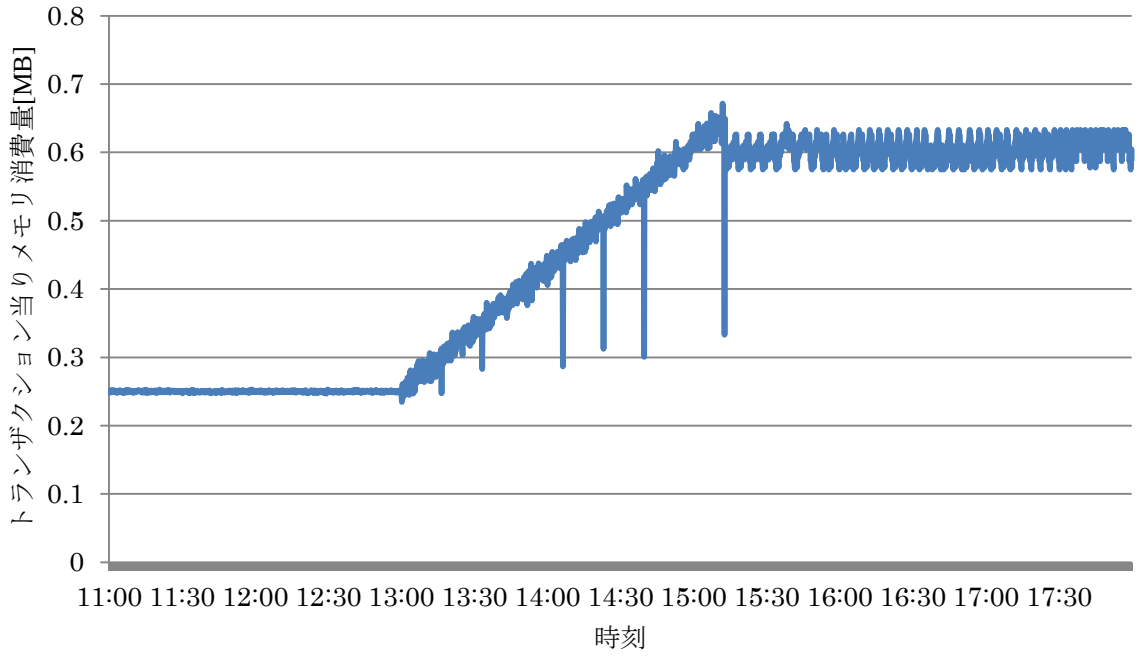


図 4-7 メモリリーク発生時のトランザクションあたりメモリ消費量

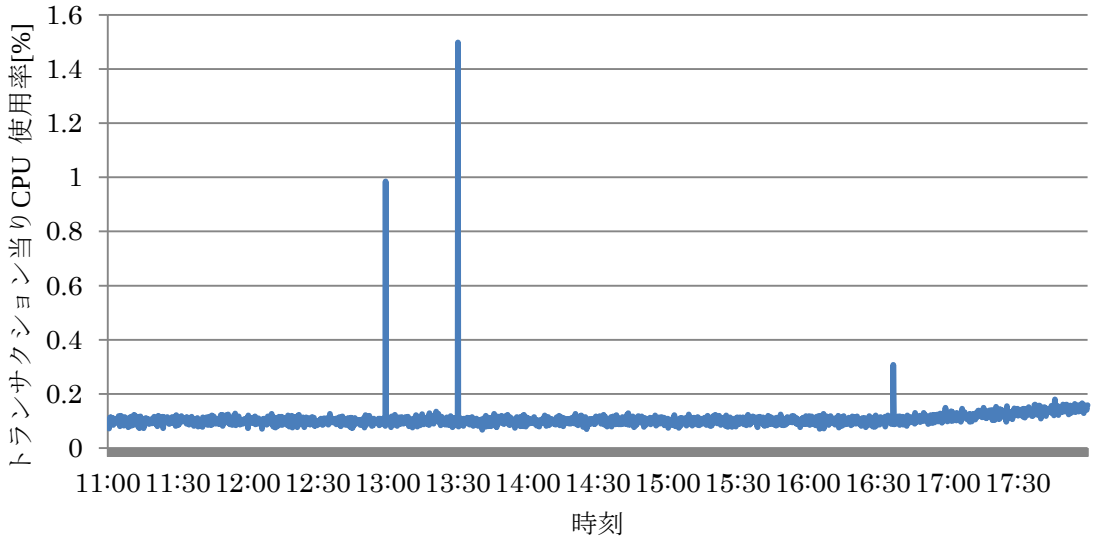


図 4-8 トランザクション当り CPU 使用率（スパイクあり）

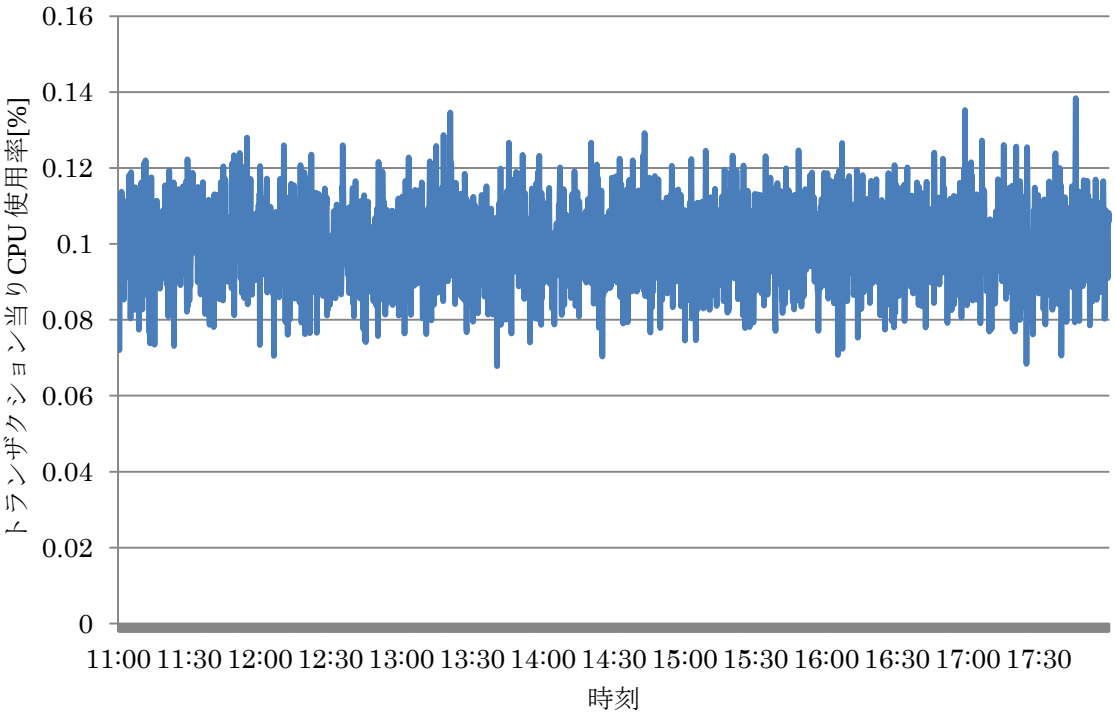


図 4-9 トランザクション当り CPU 使用率（正常時）

図 4-9 は，正常時のトランザクション当りの CPU 利用率を示している．この分布は，図 4-8 の正常時と同様の 0.1%を平均とする正規分布である．

4.5.6 実験結果

CQL のウィンドウサイズ（=W）と検定の危険率（=α）の値を変えた CQL を用意し実験を行った．図 4-10 に CQL の例を示す．図 4-10 の CQL 例は，メモリ消費量についてウィンドウサイズ 10 で，危険率 0.5%で片側検定する場合の CQL を示している．ここでウィンドウサイズの変更によって影響を受けるのは 3 つめの MOVING_AVG，5 つめの TEMP2_VARIANCE の 2 つの CQL のウィンドウ指定の[ROWS 10]の 10 の部分である．又，また，危険率を変更することによって影響するのは，7 つめの DECISION1 という CQL の中で使われている 3.250 という値だけである．この値は，数表を用いて設定された危険率から求めた値である．また，検定する対象となる設計基準値（この場合には 0.25MB）によって影響を受けるのも，やはり 7 つめの DECISION1 で使われている 0.25 という値だけである．ストリーム名称を除く CQL の他の部分は，検定方法が同じであれば不変なので，用意された CQL のテンプレートパラメタを与えて CQL を生成することは容易である．

図 4-7 のデータに対して，異常発生から異常検知を行うまでの時間を計測した結果を表 4-2 に示す．この結果から W を大きくとることにより異常検知までの時間が短縮されることがわかる．また，α が大きいほど早期に異常を検知することがわかる．

図 4-8 のデータに対して，それぞれ 3 つのスパイクを検知できるかどうかを実験により確認した．最初のスパイクに対する検知の結果を表 4-3 に示す．2 つめのスパイクは，表 4-3 に示す α と W の全ての組み合わせで検知に成功した．一方，3 つめのスパイクはいずれの組み合わせでも検知に失敗している．この結果から，スパイクの検知においては，危険率は関係せず，ウィンドウサイズは小さすぎても大きすぎても検知できないことがわかる．

表 4-4 は，図 4-9 に示す正常状態のデータに対して異常と報告する誤報の発生状況を示

```
// 入力ストリーム定義
register stream STREAM_IN(
t TIMESTAMP,
login INTEGER,
transaction INTEGER,
one_memory DOUBLE,
);
//性能改善のためのクエリ
register query Q_IN ISTREAM(
select * from STREAM_IN[NOW]
);
//1トランザクションあたりのメモリ使用量移動平均(W=10の場合)
register query MOVING_AVG ISTREAM(
select MAX(Q_IN.t) as t,
AVG(Q_IN.one_memory) as one_memory from Q_IN[ROWS 10]
);
//検定式の左辺(1次計算－分散の算出)
register query TEMP1_VARIANCE ISTREAM(
select Q_IN.one_memory*Q_IN.one_memory/9 as one_memory from Q_IN[NOW]
);
//検定式の左辺(2次計算－分散の算出)
register query TEMP2_VARIANCE ISTREAM(
select SUM(TEMP1_VARIANCE.one_memory) as one_memory,
from TEMP1_VARIANCE[ROWS 10]
);
//検定式の左辺(3次計算－分散の算出)
register query VARIANCE ISTREAM(
select MOVING_AVG.t as t, TEMP2_VARIANCE.one_memory
from TEMP2_VARIANCE[NOW],MOVING_AVG[NOW]
);
//メモリ使用量に対する検定式の判定(片側検定　平均より大きい場合)
register query DECISION1 ISTREAM(
select MOVING_AVG.t as t, 0.5 as status from MOVING_AVG[NOW],VARIANCE[NOW]
where 3.250*3.250*VARIANCE.one_memory<10*(MOVING_AVG.one_memory-
0.25)*(MOVING_AVG.one_memory-0.25) and MOVING_AVG.one_memory > 0.25
);
```

図 4-10 実験で用いた CQL の例

している．この結果から，危険率を大きく設定しすぎると誤報が発生することがわかる．また，ウィンドウサイズを大きくすることで誤報の回数が増加する傾向が見えるが，40 程度で頭打ちとなっており，ウィンドウサイズは誤報発生に関しては大きな要因となっていない．

表 4-2 異常発生検知時間（単位，分:秒）

α	W					
	5	10	20	30	40	100
0.100		16:20	07:10	05:50	06:10	05:06.
0.050		20:00	08:50	07:00	06:50	06:50
0.025		26:30	11:10	08:10	07:20	07:20
0.010		35:30	13:10	10:00	08:10	08:10
0.005		45:00	17:50	10:30	08:50	08:40

表 4-3 スパイク発生検知

α	W								
	5	8	9	10	20	30	40	100	500
0.100	×	×	○	○	○	○	○	○	×
0.050	×	×	○	○	○	○	○	○	×
0.025	×	×	○	○	○	○	○	○	×
0.010	×	×	○	○	○	○	○	○	×
0.005	×	×	○	○	○	○	○	○	×

表 4-4 誤報発生数

α	W												
	20	21	22	23	24	25	26	27	28	29	30	40	100
0.100	0	8	8	8	8	8	8	9	10	13	15	27	14
0.050	0	0	0	0	0	0	0	0	0	0	0	1	0
0.025	0	0	0	0	0	0	0	0	0	0	0	0	0
0.010	0	0	0	0	0	0	0	0	0	0	0	0	0
0.005	0	0	0	0	0	0	0	0	0	0	0	0	0

4.5.7 実験結果の考察

表 4-2 から W を大きく，α を大きくとるほど異常事態発生の検知時間は短くなる．W は 40 程度以上では，検知時間はあまり変わらなくなる．4.5.4 節でのストリーム DB のメモリ消費量に関する考察と合わせて考えると，W を 40 程度とするのが適当である．

表 4-3 から W を 9 より大きくすれば大きなスパイクの検知ができる．また，逆に W=500 では検知できないことから，W を大きく取りすぎると，解析対象データが平均化されスパイクが逆に検知されなくなる．13:30 の最大のスパイク発生時点周辺の，CPU 使用率の移動平均の状況を図 4-11 に示す．W=500 では，平均化されスパイク発生時点周辺でもほとんど変化がみられないことがわかる．図 4-12 に同じ時間帯の CPU 使用率の分散の変化を示す．W を変化させたときの移動平均と分散の状況は，当然ながら同様の傾向を示している．実験では，スパイクは，（4-9）の右側の不等式を変形した，

$$z_{\frac{\alpha}{2}}{}^2 s^2 > W\left(\overline{X} - m\right)^2 \qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (4\text{-}12)$$

が成立しないときに発生が検知される．W が小さいとき，すなわち t 分布の自由度が小さいときには，$z_{\alpha/2}$ の値は，大きな値となる．そのため，W の値はある程度の大きさが必要である．今回の実験では，9 以上の値が必要ということなる．

表 4-4 から，α の値が 0.1（10%）では，正常時でも発生する実測値のぶれに対する誤報がかなり発生するが，α を 0.050（5%）程度より小さく取ることで，誤報はほとんどなくなる．α の値が大きいと $z_{\alpha/2}$ の値が小さいため，わずかな実測値のぶれに対しても，式（4-12）が成立しなくなることがその理由である．

上記のように，図 4-7 に示す異常と図 4-8 に示す異常のように，最適な W の値が異なる場合がある．ストリーム DB のメモリ消費量を削減する観点では，同じ CQL で監視することで CQL の数を低く抑えることが望ましいが，高い精度が要求される場合には，検知する異常のタイプ毎に別々の CQL を用意してもよい．

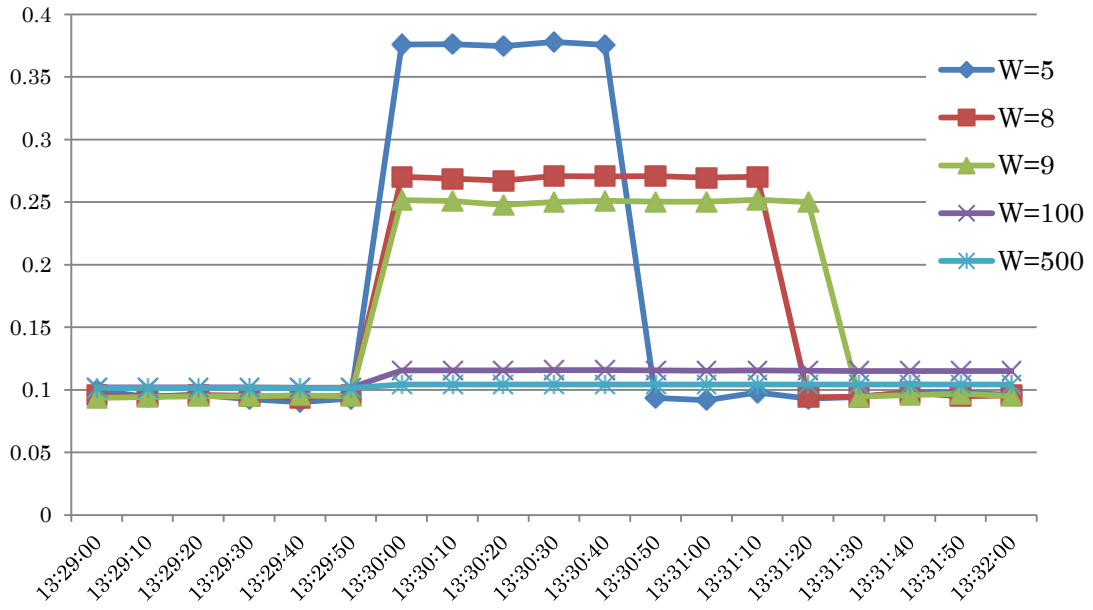


図 4-11 スパイク発生時の移動平均

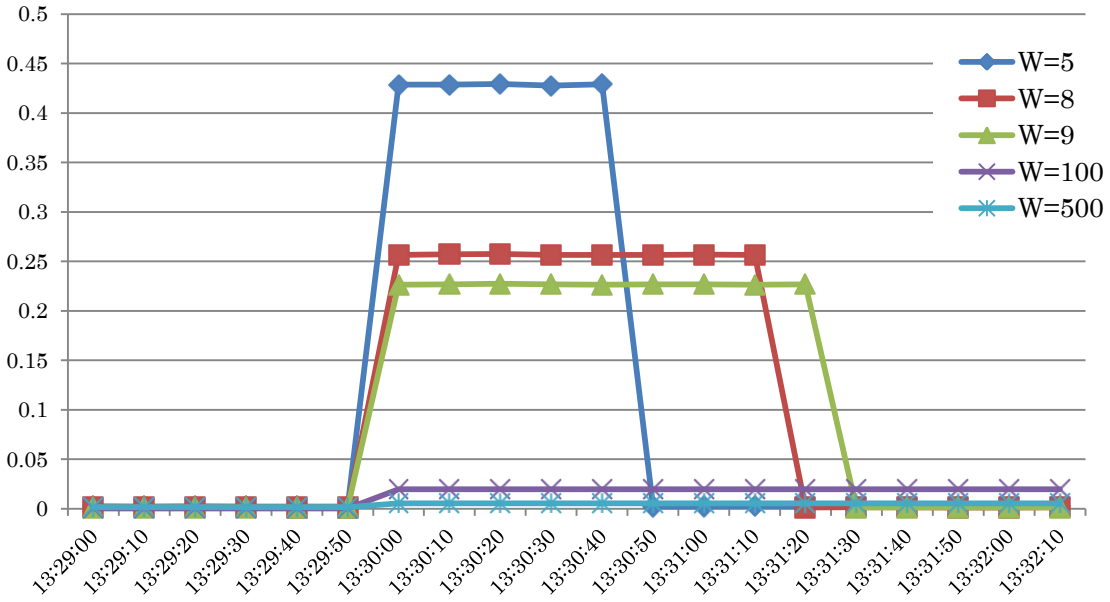


図 4-12 スパイク発生時の分散

4.5.8 関連研究との比較

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[33]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[73][74]．

これらの関連研究と本章の提案方式を比較すると，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章の提案方式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，本章の結果の適用範囲が限定されることはない．

情報システムのログ情報を統計的に分析する試みはさまざま行われているが，リアルタイムに統計的に分析して障害の予兆を検知し運用管理に活用する手法は新しい手法であると考えられる．

4.6 結言

本章では，ログ情報のリアルタイム監視及び統計的検定にストリーム 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．特に，Web アプリケーションのアーキテクチャに基く設計基準値をベースにしたキャパシティプラニングにおいて，実測基準値と設計基準値との偏差を統計的手法に基いて検定することで，キャパンシティプラニングの想定とおりにシステムが稼働していることの確認を行う手法を提案した．さらにストリーム DB を利用して，この検定をシステム稼働時にリアルタイムに行う場合に，検定精度とストリーム DB のスケーラビリティ確保の観点で問題となる

ウィンドウサイズについて，実験により適切な設定の指針を示した．

クラウドコンピューティングの普及に伴い，クラウド上で稼働するアプリケーションの監視の重要性が増している．本章で提案した方式により，監視の精度をコストパフォーマンスを考慮しながら定量的に制御できるようになる．今後，本方式のクラウド監視サービスへの適用が期待できる．

本章では, キャパシティプラニング時に想定した設計基準値との偏差を検定しているが, 今後，正常状態時の基準値の分布をストリーム DB で測定し，それとの偏差を検定するといった方式を発展形として検討する．また，本章では提案した手法を，Web アプリケーションの稼働監視に適用したが，情報システムに限らずセンサ等で稼働情報を取得できるシステムに本手法の適用を拡大していくために，適用場面毎に適切な検定方式を選択することと, その検定方式をストリーム DB を利用して実装するための CQL 自動生成方式の整備, 必要なストリーム DB エンジンの機能拡張の検討が今後の課題である．

第 5 章
情報システム運用手順パターンのデータセンタ省電力運用への適用拡大

5.1 緒言

本章では，第 2 章で抽出した情報システムの運用手順のパターンを，データセンタの省電力運用に拡張する方法について論じる．

環境に対する認識の高まり[75]により「グリーン IT」が重要なキーワードとなっている[76]．「グリーン IT」には，2 つの異なった観点がある．一つは IT リソース自体のエネルギー消費の減少を目指す greening of IT[77] である．クラウドコンピューティングサービスを提供する巨大なデータセンタにおいては, サーバ電力消費量の費用が深刻な問題となっている．大規模データセンタにおいては，サーバのエネルギー消費と性能が比例するとの報告[784]もある．このような状況により，greening of IT は IT ガバナンスにおいても重要な観点となっている[79]．

「グリーン IT」のもう一つの観点は IT の活用による様々な活動に伴うエネルギー消費量の削減である．例として，テレビ会議システムの利用による移動の削減，書類を使った業務を電子化することによる紙の消費量の削減，工業製品の設計過程でのコンピュータ・シミュレーションの活用による実物による実験の削減，高度交通情報システムにより交通渋滞を減少させて燃料消費量を削減する，などが挙げられる．これは greening through IT[80]と呼ばれる．スマート・グリッド[81]も greening through IT の一例である．

企業情報システムにおいて，適切なシステム管理は情報システムの信頼性と効率性の確保のために不可欠である．適切なシステム管理なしには，情報システムは企業内外のユーザに対して期待された機能を提供することはできず，情報システムの開発への投資は期待した効果をもたらすことができない．システム管理作業を効率化するために，システム管理作業の全てあるいは一部を自動化するためのシステム管理ツールが広く使用されている．一方，近年の「グリーン IT」のニーズの高まりにより， 情報システムのエネルギー消費の管理はシステム管理の重要な一面となっている．従来は，情報システムの運用管理の対象は IT 機器及びソフトウェアであり，省電力運用の対象は電源機器，空調機器，照明などの設備が対象であり，全く異なる部署が管理していることから，個別に行われている場合が

多い．商業施設やオフィス等を対象とした快適さと省エネルギーを両立させるための運用自動化フレームワークの提案[34]もなされている．省電力化のニーズがより高まる中で，情報システムの状況に合わせてデータセンタ内の付帯設備を制御する，あるいはデータセンタの状況に合わせて情報システムの運用を制御することで，さらに省電力化を図る方法が研究されており，成果が上がってきているが，情報システムの運用と設備運用を統一的にとらえて省電力を考えるための枠組みはまだ存在しない．

情報システムの運用と設備運用を統一的にとらえる枠組みの構築に向けて，本章では，システム管理ツールとシステム運用管理プロセスの「グリーン IT」への適用方法を，第 2 章で抽出した運用管理手順のパターンごとに省電力運用シナリオをパターンのユースケースとして示すことで例示的に示す．また，運用管理手順を定式化して記述し，ITIL で提唱されている CMS と組合わせることで，運用管理手順の省電力効果の定量評価を行う方法を提案し，事例をベースに有効性を検証する．さらに，省電力効果の定量的評価を行うことで課題を明確化し，改善計画を立案した例を示す．

本章の構成は以下のとおりである．5.2 節でシステム運用管理とグリーン IT の関係，関連する技術について概説する．5.3 節では，5.4 節で運用管理手順パターンに従って省電力運用のユースケースを記述するための記述方法を提案する．5.4 節では，5.3 節で提案した記述方法に従って，省電力運用に適用したユースケースを記述し，省電力運用を十分記述できることを確認する．5.5 節では実際の情報システムの事例を使って，提案した運用手順記述方式を用いた省電力効果算出例を示す．

5.2 システム運用管理とグリーン IT

5.2.1 システム管理プロセスとツールの範囲

情報システムの管理に関するベストプラクティス集である ITIL では業務システムのライフサイクルを「要件」，「設計」，「構築」，「展開」，「運用」，「最適化」の 6 つのステージで定義している．通常，ITIL で定義されている「運用」段階はアプリケーションのライフサイクルの 6 つのステージの中で最も長い．そのため，IT のグリーン化のためには，「運用」段階の期間におけるエネルギー消費量を減らすのが最も効果的である．そのため，システム運用管理操作の一部としてエネルギー消費戦略を実装するのが妥当である．

システム運用管理操作の一部あるいは全てを自動化するために利用可能なツールが広く市場で入手可能である．これらのツールは，(1) 監視ツール：監視対象資源の状態とイベントを監視あるいは，それらに関するログデータの収集を行う，(2) 制御ツール：管理対

象資源の状態変更動作を実行，（3）調整ツール：監視ツールの情報を基に行うべき行動を決定して，適切な制御ツールを呼び出す，の 3 種類に分類できる．

5.2.2 関連する技術

サーバシステムの電力消費量を抑えるために利用可能な技術をまとめた文献[82]によれば，局所的な技術だけでは大規模なデータセンターにおける電力消費量管理の課題には不十分であり，全体的なアプローチが必要であると述べている．本節では，システム運用管理の観点からこれらの技術を再整理する．電力消費量の管理をシステム運用管理の一部としてとらえることで，必要な関連技術が広範囲に適用されると考えられる．

地理的に離れた場所に散在する管理対象資源を一箇所の集中管理コンソールから管理することで，大量の IT 資源を効率的に管理するためには，遠隔での監視と制御を可能とする技術が重要である．これを可能とするために IT 資源の分散管理を Web サービス技術を利用して行う仕様である WS-Management[83]の標準化活動が DMTF（Distributed Management Task Force）で推進されている．

熟練したシステム運用管理技術者がデータセンタで最も貴重なリソースであるため，システム運用管理コスト削減のためには，システム運用管理操作の自動化が不可欠である．システム運用管理の自動化の進展に向けた産業界の活動が報告されているが[84] [85]，現在の状況はまだ不十分な状況である[17]．「グリーン IT」の観点でみても，遠隔かつ自動で運用管理操作を行うことで技術者の移動などの人間系関連のエネルギー消費量を劇的に削減することができる．サーバ統合やクラウドコンピューティングなどの IT のトレンドにより，データセンタで管理すべき IT 資源の数が増大しているが，データセンタの巨大化すればするほどシステム運用管理の自動化もより重要となる．

5.3 運用手順のパターンと CMS を前提とした運用手順記述

本節では，2.3 節で示した運用手順のパターンと ITIL で定義されている構成管理システム（CMS）を前提とした運用手順の記述方法について述べる．

5.3.1 用語の定義

構成管理システムの説明に必要な用語の定義を，ITIL の用語集から下記のように引用する．

① 構成アイテム（CI: Configuration Item）

IT サービスの提供のために管理する必要がある，あらゆるコンポーネント．各 CI に関する情報は, 構成管理によって構成管理システム内の構成レコードに記録され, そのライフサイクルを通して維持される．CI は変更管理によってコントロールされる．一般に CI には，IT サービス，ハードウェア，ソフトウェア，建物，人材，およびプロセス文書や SLA などの正式文書が含まれる．

② 構成管理システム（CMS: Configuration Management System）

IT サービス・プロバイダの構成データの管理に使用される，一連のツールまたはデータベース．CMS には，インシデント，問題，既知のエラー，変更，リリースに関する情報も含まれる．また，従業員，サプライヤ，場所，事業部門，顧客，ユーザに関するデータも含まれる場合がある．CMS には，すべての構成アイテムおよびそれらの関係に関するデータを収集，保管，管理，更新，提示するためのツールも含まれる. CMS は, 構成管理によって維持され, すべての IT サービスマネジメント・プロセスに使用される．

構成管理システム（CMS）は構成管理データベース（CMDB（Configuration Management Database））と呼ばれることもある．

5.3.2 運用手順記述形式

本節で示す記述形式の目的は，具体的に運用管理ツールの入力となるような運用手順記述形式ではなく，具体的な運用手順記述形式を設計する際に必要となる記述要素を抽象レベルで定義することである．そのため，運用手順記述は，擬似コード形式で与える．具体的な運用手順記述形式は，記述要素を例えば XML（Extensible Markup Language）形式で表現することになる．

本章で示す運用手順記述形式は運用の自動化を目的としたものであり，運用の自動化を実現するためには，5.3.1 節に定義を示した ITIL の CMS の存在が必須であるため，これを前提として考える．

構成アイテムの定義によると，運用手順自身も構成アイテムとして CMS で管理されるべきものである．CMS で管理されるためには，CMS 内で一意に識別できる識別子が必要である．従って，運用手順形式はそれ自身の識別子を持つ必要がある．

運用手順を 2.3 節で示した運用手順パターンを前提に運用手順を管理するため，その運用手順がどのパターンに属するかを示す情報を持つ必要がある．

運用手順がどのパターンに属すかが決まると，運用手順記述に必要な作業単位が特定で

```
<Operation> ::= <Operation ID><Pattern ID><Operation-body>
<Operation-body> ::= <Event-Detection> <Action-body>
<Action-body] ::= <Condition-Check-list> <Judgment>
                    { <Condition> <Action-list>}+ |
                     <Action-list>
<Condition-Check-list> ::= <Condition-Check> {<Condition-Check-list>}
<Action-list>  ::= {<Action>}
<Operation ID> ::= Operation : operation name
<Pattern ID> ::=  Pattern: pattern name
<Event-Detection> ::= Event: <Target> event detection description
<Condition-Check> ::= Condition: <Target> condition check description
<Judgment> ::= Judgment
<Condition> ::= When: check result description
<Action> ::= Action: <Target> action description
<Target> ::= Target: target description
```

図 5-1　運用手順記述形式の BNF 記法による表現

きる．各作業単位の作業内容，作業の対象となる IT リソースの組み合わせで記述する．ここで，各作業単位の作業内容と，作業の対象となる IT リソースもやはり全て CMS で管理され識別子を持つとする．

上記を考慮した運用手順記述形式を BNF（Backus-Naur Form）記法で表現したものを図 5-1 に示す．

5.4 運用管理手順パターンに基づく電力消費削減を意識した運用手順

以下，第 2 章に示した運用管理手順パターン毎に，電力消費削減を意識した運用手順について説明する．

5.4.1 定期的定型運用のユースケース

定期的定型運用パターンはタイマイベントによって起動されるシステム運用管理操作に適用される場合が多い．この場合は通常，運用管理操作は状態確認および判断なしに実行される．このパターンの運用管理手順の最も簡単な例は，システムの起動・終了である．電力消費削減運用のユースケースとして，遠隔電源制御機能を利用した次のようなユースケースが考えられる．

Operation : shutdown
Pattern : Periodic Customary
Event : Target : timer at 11 p.m. on business days
Action : Target : Server123 turn-off

Operation : wakeup
Pattern : Periodic Customary
Event : Target : timer at 6 a.m. on business days
Action : Target : Server123 turn-on

図 5-2: 定期的定形運用パターンの例

ユースケース 1: 起動時間帯が決まっているシステムに対して，システム起動前にタイマイベントに基づいてサーバの電源を遠隔でオンにし，システム終了後に遠隔で電源オフすることで，サーバの電源を必要な時だけオンにする．

この運用手順を，5.3 節に示した運用手順記述形式で記述した例を図 5-2 に示す．
同様のシナリオをクライアント PC に適用すると以下のユースケースが得られる．

ユースケース 2: 設定された時間にユーザ応答を要求する警告メッセージをクライアント PC に送信し，設定された時間の後に，ユーザからの応答が無い場合に利用が終了しているとみなし PC の電源を遠隔でオフにする．

ユースケース 1 は，システムが限られた時間帯だけで動作する場合に適用できるが，ノンストップ運転しているシステムでもシステムの負荷が時間帯によって変化し，システムが複数のサーバで運用されていて，負荷に応じてサーバ台数の調整が可能であれば，必要なサーバにだけ電源を入れることで，消費電力削減を図るユースケースを考えることができる．電子メール等のコラボレーションツールは，営業時間帯とそれ以外の時間帯で負荷が大きく変化するため，このような運用が可能なシステムの例である．

ユースケース 3: 事前に負荷が軽い時間帯がわかっている場合に，そのタイマ起動によりその時間帯にシステム構成を変更し不要なサーバの電源をオフにし，負荷が高い時間帯には逆に休止させていたサーバの電源をオンにし，システム構成を変更してサーバ台数を増加させる．

Operation : scheduled maintenance
Pattern : Non-periodic Customary
Event : Target : timer at specified time during turned off period
Action : Target : Server123 turn-on
Action : Target : Server123 version up of applications
Action : Target : Server 123 turn-off

図 5-3　不定期定形運用の例

5.4.2 不定期定型運用のユースケース

不定期定形運用のパターンは事前に定義されているイベント発生の自動検出，あるいはオペレータ介入により開始されるシステム運用管理手順に主として適用される．OS やミドルウェアのリビジョンアップや，セキュリティアップデートの適用などの保守作業はこのパターンの手順の典型例である．

このシナリオを以前の既に示したユースケースに応用することで以下のユースケースが得られる．

ユースケース 4: 業務休止中で電源オフしているサーバを, 保守作業の直前に電源オンし，保守作業終了後に電源オフする．電源操作と保守作業はソフトウェア制御により遠隔実行する．

ユースケース 4 の運用手順を 5.3 節で示した運用手順記述形式で記述した例を図 5-3 に示す．

ユースケース 5: 電源オフ状態のクライアント PC を保守作業の直前に電源オンし，保守作業終了後電源オフする．電源操作と保守作業はソフトウェア制御により遠隔実行する．

ユースケース 6: ユースケース 3 を適用可能なシステムにおいて，負荷の軽い時間帯に保守作業を計画し，電源オフとなっているサーバを電源オンし保守作業を実施し，完了したサーバを業務で使用するようにシステム構成を変更し，業務で使用していたサーバに対して保守作業を実施し，完了後電源オフにする．

ユースケース 6 は，ローリングアップグレード[86]と呼ばれる，業務やサービスを停止せずに保守作業を実施するために使われる手法である．業務の負荷に応じてサーバを縮退運転する運用が前提となる．図 5-4 の 1.は保守作業を実施する前の状態であり，負荷が小さい状態で電源がオフとなっているサーバが存在している．図 5-4 の 2.は第 1 フェーズを示しており，保守作業実施のために電源オフとなっているサーバに電源を投入し，保守作業を実施する．図 5-4 の 3.は第 2 フェーズを示しており，保守作業が完了したサーバで業務を実行するように構成変更を行い，これまで業務で使用していたサーバを保守作業できる状況にする．図 5-4 の 4.は作業終了後の状況を示しており，全てのサーバの保守作業が終了しており，業務で使用していないサーバの電源を切断した状態である．この方法により，業務を停止させることなく，また保守作業の準備のために業務で不要なサーバに予め電源を投入しておくことなく保守作業を実施することができ，サービスレベルを保持しながら省電力運用を実現できる．

5.4.3 注意喚起のユースケース

注意喚起パターンは，システムの状態の変化やイベント発生を報告するために用いられ

図 5-4 ユースケース 6 の実行例

る．状態監視，イベント監視が自動的に監視ツールによって行える場合にこのパターンの運用管理手順が使用される．すぐに対応を行う必要が無い場合，あるいは自動的に対応処理を行うことが困難であり，オペレータが対応する必要がある場合にこのパターンの運用管理手順が使用される．

家庭でのエネルギー消費量を示すことが，エネルギー消費行動を改善することにつながるという研究結果[87]が報告されている．オフィスの PC についても同様のことが期待でき，次のようなユースケースが得られる．

ユースケース 7: オフィスに設置されている PC の省電力対策状況を定期的に収集し結果をマネージャとユーザに送付する．

省電力対策には，使用していない時にモニターやとハードディスクを停止させるといったことが考えられる．一定時間操作されていない場合に PC を待機モードに移行し消費電力を低減する．このための設定変更が遠隔で行える場合には，その設定のためにユースケース 5 を適用することができる．

大規模データセンタの電力消費に関しては，サーバ，ネットワーク機器，ストレージ装置のようなハードウェア機器の電力消費に加え，機器を冷却するに必要な電力消費についても考慮する必要がある[88]．先進的なデータセンタでは，サーバ設置場所に温度センサと湿度センサを装備し，システム運用管理ツールと組合わせた管理を行っている．このような場合には次のユースケースが適用できる．

ユースケース 8: 温度や湿度などの環境センサーの値が設定されている閾値を超えた場合にオペレータに通知を行う．

5.4.4 障害処置のユースケース

障害処置パターンは対処が必要なイベントを検出し，必要な状態の確認を行い，その結果に基き適切な処置を選択し，その実行を行うシステム運用管理手順に適用される．ユースケース 3 では，時間帯によって負荷が予測可能であることを前提としているが，システムの負荷が監視可能であり，その結果から負荷の予測が可能である場合には，障害処置パターンを適用することができる．この場合には，以下に掲げるようなユースケースとなる．

ユースケース 9: システムの負荷を監視し，負荷が閾値以下の状態が継続すると予測され

る場合に，過剰なサーバを使用しないようにシステム構成を変更し，使用しないサーバの電源をオフにする．負荷の監視状態から負荷が増加することが予測される状況となった場合には，逆に休止状態のサーバの電源をオンにし，利用できるようにシステム構成を変更する．

ユースケース 9 の運用手順を 5.3 節で示した運用手順記述形式で記述した例を図 5-5 に示す．ユースケース 9 において，環境センサーの値が異常値となった場合の対処が自動化できる場合には，障害処置パターンを適用することができる．自動化可能な対処の例として，温度が高い場所のサーバから温度が低い場所のサーバに負荷を移動して，前者のサーバの電源をオフにするといった処理がある．サーバ仮想化技術の発達により，サービスを中断せずに，実行中のワークロードを別のサーバに移動することができる[89]．このシナリ

```
Operation : Dynamic Workload Management -add servers
Pattern : Error Recovery
Event : Target : Workload-Sensor-for-the-system
        when the workload reaches over upper limit
Condition: Target: Server-pool availability of servers
Judgment:
When: Available
   Action : Target : Server-pool Provisioning of servers available
   Action : Target : in-use-Servers-and-newly-added-servers
                     Reconfigure to utilize newly added servers
When: Not-available
   Action : Target : System-Operator
            Notify that workload exceeds the limit

Operation : Dynamic Workload Management-delete servers
Pattern : Error Recovery
Event : Target : Workload-Sensor-for-the-system
        when the workload reaches under lower limit
Condition: Target: Number-of-servers-used-for-the-system
Judgment:
When:
  Over-Lower-Limit
   Action : Target : Servers-used-by-the-system
                     Reconfigure to reduce number of servers
    Action : Target : Removed-servers
                     Turn off
When:
    Lower-Limit
```

図 5-5 障害処置パターンの運用手順の例

オを効果的に実装するためには，サーバの正確な物理的位置の管理が必要となる．以上をまとめると，次のようなユースケースが得られる．

ユースケース 10: サーバ設置場所のある領域の温度が事前に定義された閾値を超過した場合に，高温の領域のサーバから低温の領域のサーバに負荷を移し，高温の領域のサーバの電源をオフにする．

図 5-6 にこのユースケースの適用例を示す．この例では，イベントとして温度の異常を通知され，オペレータが監視ツールを使ってシステムの状態を確認し，とるべき処置を決定し制御ツールを用いて処置を行う．

5.4.5 ユースケースのまとめ

表 5-1 にこれまでに挙げた 10 の消費電力削減のためのユースケースをパターン毎にまとめる．表 5-2 に各パターンのシステム運用管理手順を自動化するための前提条件をまとめる．どのパターンでも，順番に複数の処理を実行するための調整ツールが必要となる．電力消費削減の手順で必要な機能は，遠隔操作が可能でなければならない．定期的定形運用のパターンにおいては，カレンダー管理などのスケジューリング管理システムの存在が前提となる．表 5-1 と表 5-2 を用いて，データセンターにおいて現在のシステム運用管理プロセスとシステム運用管理ツールを活用してどのような消費電力削減手順を実装することができるかを検討することができる．また，データセンタの電力消費削減を更に推進するために必要となる，システム運用管理プロセスの改善及びシステム運用管理ツールを検討する際に利用することができる．

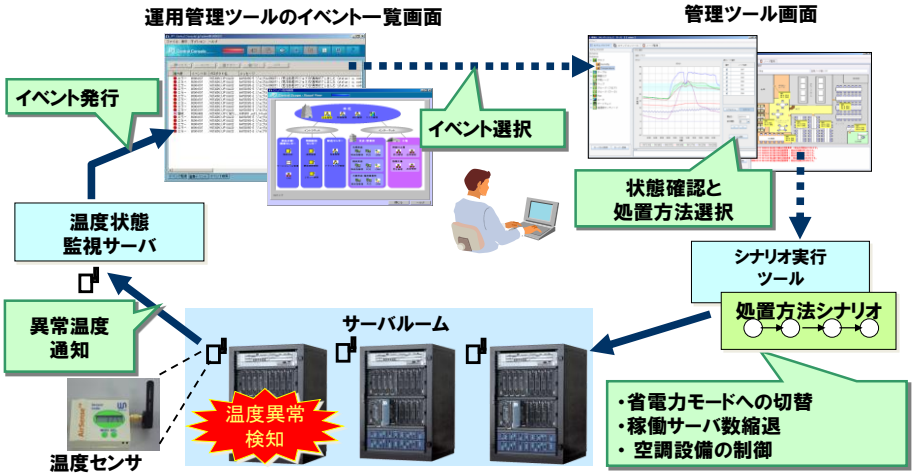


図 5-6 ユースケース 10 の例

表 5-1 ユースケースのまとめ

パターン	ユースケース
定期的定常運用	ユースケース 1: スケジュールに従ったサーバの遠隔電源操作
	ユースケース 2: 警告メッセージ発行を伴うスケジュールに従った PC の遠隔電源操作
	ユースケース 3: スケジュールに従った電源投入サーバ数の増減
不定期定常運用	ユースケース 4: 保守作業実施のためのサーバの遠隔電源操作
	ユースケース 5: 保守作業実施のための PC の遠隔電源操作
	ユースケース 6: 電源投入サーバ数を減少させるローリングアップデート
注意喚起	ユースケース 7: オフィス内の省電力ポリシー適用監査の報告
	ユースケース 8: サーバルームの温湿度変化の報告
障害処置	ユースケース 9: 負荷監視の結果に基く電源投入サーバ数の増減
	ユースケース 10: 高温エリアのサーバから低温エリアのサーバへの負荷移動及び前者サーバの電源オフ

表 5-2 運用手順の前提条件

パターン	前提条件
全パターン	複数操作を連携する調整ツールが存在する．． 省電力操作を含む操作が遠隔から実行できる.
定期的定常運用	定期的操作の仕掛けが確立している．（カレンダ管理など).
不定期定常運用	不定期操作についての手順が確立している.
注意喚起	イベントが調整ツールで処理可能である. 注意喚起に対応する処理が確立している.
障害処置	イベントが調整ツールで処理可能である．． 自動的処置およびオペレータ介入の条件が確立している．．

5.5 電力消費量削減評価と分析の例

5.5.1 評価で用いる事例

大企業のコラボレーションツールシステムを事例として選択する．ユースケース 3 を適用して電力消費量削減を評価する．表 5-3 に評価対象事例の関係するデータを示す．表 5-3 のデータは規模等は実際の企業システムのデータに基づいているが，全く同一ではない．

この会社はグローバルにビジネス展開しているため，コラボレーションツールのユーザは全世界に散らばっている．しかし， 90%以上のユーザは日本を中心とする東アジアで勤務している．容易に予想できるように，コラボレーションツールの負荷は営業時間中は高いがそれ以外の時間帯は負荷が低くなる．ユーザが全世界に一様に分布していないため，このシステムの負荷は時間帯によって変動する．そのため，ユースケース 3 を適用できる．すなわち東アジアの営業時間以外の時間帯でサーバ数を減少させることができる．図 5-7 にログインユーザ数およびメールの送受信数の日本時間の一日の間の変動を示す．グラフは，ピーク値を 100 とする相対値を示す．図 5-7 から，このシステムの日本時間 22:00 から 8:00 までの 10 時間の負荷は，ピーク時の 25%未満であることがわかる．

オフピーク時にスケールイン・アウト可能なサーバの稼働台数を減らし，電源オフする運用手順により，省電力運用を実施する場合を考える．この運用手順を 5.3 節で示した運用手順記述形式で記述した例を図 5-8 に示す．

表 5-3 評価対象システムの概要

アプリケーションの種類	大企業における，e-mail，スケジュール管理，掲示板機能等を含むコラボレーションツール
登録ユーザ数	約 30 万人
ユーザの地理的分布	90%以上が極東地域
サーバ数	Web/アプリケーションサーバ: 30 DB サーバ: 5 その他サーバ: 50
サーバの電力消費量(kW/台)	Web/アプリケーションサーバ: 0.3 kW DB サーバ: 0.7 kW その他サーバ: 30 kW
オフピーク時の負荷	オフピーク時の負荷は通常時の 25%未満

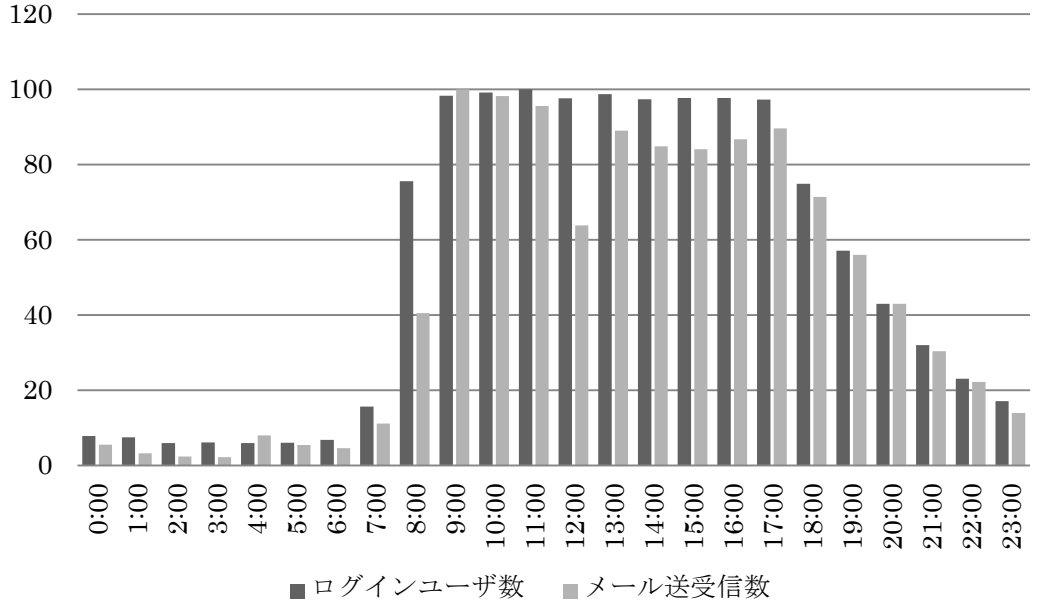


図 5-7 コラボレーションツールの一日の中の時間毎の負荷変動

5.5.2 評価結果と分析

情報システムの省エネルギー効率を算定する方法として，ライフサイクル全体を評価する SI-LCA（System Integration – Life Cycle Assessment）[90]といった手法があるが，本節では単純に時間当たりの稼動サーバ数と 1 サーバあたりの電力消費量から時間単位の電力消費量を算出し評価する．

Operation : Collaboration system management -add servers
Pattern : Periodic Compulsory
Event : Target : Timer
at 8 a.m. on business day
Action : Target : Turned-off-servers
Turn on
Action : Target: Servers-for-this-system
Reconfigure to utilize turned on servers

Operation : Collaboration system management-delete servers
Pattern : Periodic Compulsory
Event : Target : Timer
at 10 p.m. on business day
Action : Target : Servers-used-by-the-system
Reconfigure to reduce number of servers by 22
Action : Target : Removed-servers
Turn off

図 5-8 評価に用いた運用管理手順

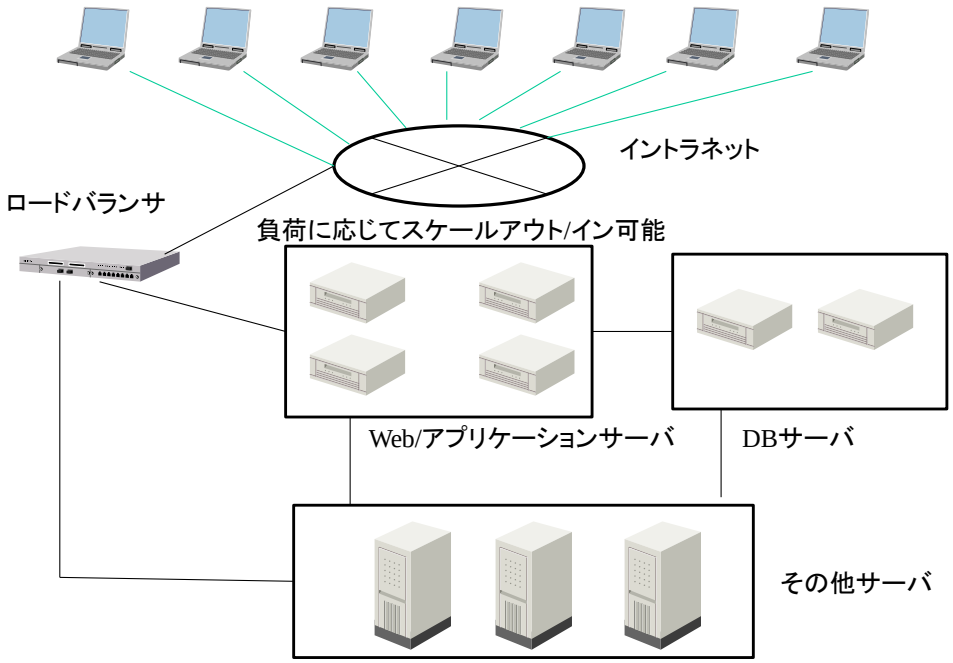


図 5-9 モデルケースのサーバ構成

コラボレーションツールは論理的には典型的な Web3 階層アプリケーションである．しかしながら，同一の物理サーバ上に Web サーバとアプリケーションサーバの機能の両方を配置しており，物理的には 2 階層となっている．Web3 階層アプリケーションでは，適切なアーキテクチャで開発することで Web サーバ及びアプリケーションサーバをスケーラブルにすることが可能であるが，データベース層をスケーラブルにするのは困難である．そのため，Web サーバとアプリケーションサーバは，負荷に応じて数を増減させることができるが，負荷が変動してもデータベースサーバの数は増減させることができないということになる．Web サーバとアプリケーションサーバについては，使用していないサーバの電源をオフにする，あるいは他のアプリケーションで利用することができる．今回は，使っていないサーバは電源をオフにするとして評価を行う．図 5-9 にコラボレーションツールのサーバ構成を示す．Web サーバ，アプリケーションサーバ，データベースサーバの他に負荷によらず常に必要なサーバ群が存在する．このシステムの負荷は東アジアの営業時間帯が最大となる．日本時間の 8:00 から 22:00 までの時間帯を営業時間帯と定義する．図 5-7 に示すように，営業時間帯外の負荷は営業時間帯の負荷の 1/4 以下となる．そのため，営業時間帯外では，Web アプリケーションサーバの台数を営業時間構成の 1/4 とすることができる．評価においては，営業時間帯に 30 台稼動している Web/アプリケーションサー

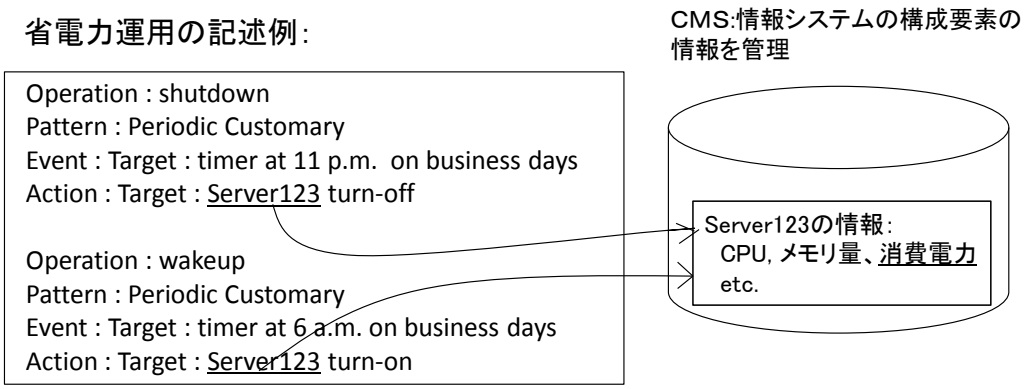


図 5-10 運用手順記述と CMS の関係

バの台数を営業時間帯外には 8 台に減少させると仮定する．1 年間の営業日は約 250 日とする．1 年を時間数に換算すると 8,760 時間 （24 時間/日*365 日/年）である．稼動するサーバ数は営業日の営業時間帯外（10 時間/日*250 日/年）及び非営業日（24 時間/日*115 日/年）の間は減少させることが可能である．合計すると減少されるサーバの稼働時間は，5,260 時間となる．Web/アプリケーションサーバに使用されているサーバの平均電力消費量に基いて，節減可能な電力量を計算する．22 サーバ*5,260 時間*0.3kW/サーバ=3 万 4716 kWh となる．これは日本の平均的家庭 8 戸分の電力消費量にほぼ等しい．

このような省電力効果の算定を行う際に，5.3 節で述べた ITIL の構成管理システム（CMS）を利用し，各サーバの電力消費量のデータを格納しておくことで，図 5-7 に示した運用管理手順と組合わせて，サーバ縮退運用による電力量削減効果を計算することができる．運用管理手順と CMS の関係を図 5-10 に示す．

この評価で想定される電力消費量は絶対量として小さなものではないが，システム全体の電力消費量との比較で考えると，全体の約 0.26%を改善しただけに止まっている．負荷によらず必要となるサーバがシステム内に数多く存在し，これらのサーバの電力消費量が多いことが原因である．より電力消費量を削減するためには，現在負荷に応じて増減することができないサーバ群を負荷に応じて増減できるようにするためのアーキテクチャ変更が必要となる．あるいはハードウェアの更新に際して固定的に必要なサーバを電力消費量の少ないもので入替えることを考慮する必要がある．コラボレーションツールの次世代を検討する際に，エネルギー効率の観点を入れるためにはこのような評価が不可欠である．

上記評価時点では負荷に応じ増減不可能だったサーバを増減可能とするコラボレーションツールのアーキテクチャの改良がその後実施されている．それを適用した構成見積りによると，表 5-3 のその他サーバの内 34 台を増減可能な Web/アプリケーションサーバ 16 台

と増減できない DB サーバ 4 台で置換え，更に前者は消費電力量の少ないハードウェアを適用できる．オフピーク時に増減可能なサーバの電源をオフにする運用を行うと，オフピーク時には 12 台のサーバの電源をオフとでき，12 サーバ*5,260 時間*0.3kW/サーバ=1 万 8936 kWh の電力を更に節約できる．元々の節約分と合わせると，システム全体の電力消費量の約 0.4%の改善となる．

この例では，アプリケーションのアーキテクチャを改善することで，どの程度の省電力効果があるかを試算したが，他に種々の施策がありうる．電力消費量の大きい旧モデルのサーバの使用が問題というケースもあるし，システムの負荷の監視が適切に行えないために縮退運転が行えるにもかかわらず，縮退運転による省電力運用が効果的に行えないなどのケースも考えられる．運用手順の記述と CMS を組合わせることで，施策を実施した場合の省電力効果をシミュレーションすることが可能となり，最も効果的な施策を選択できるようになることが期待できる．

5.6 結言

第 2 章で抽出したシステム運用管理手順のパターンに基く電力消費削減のための運用のユースケースを挙げることで，システム運用管理手順のパターンを省電力運用手順を分類し，改善方法を検討するために利用する例を示した．この例示により，現状のシステム運用管理のプロセスとツールを活用して，どのような電力消費削減運用が可能かを検討する場合に，システム運用管理手順のパターンの考え方が利用できることを示した．また，さらに消費電力を削減するためには，「イベント検知」のレベルアップが必要なのか，「アクション」のレベルアップが必要なのかなど，省電力運用の高度化に必要な施策を見通しよく立案するために，システム運用間手順のパターンの考え方が役立つことを示した．また，コラボレーションツールの事例を用いて省電力効果評価において，運用管理手順記述と CMS を活用して行えることを示した．

システム運用管理をデータセンタの省電力運用に拡張する例として，サーバルーム内の温湿度センサによる異常状態の検知をシステム運用管理と結びつけ，サーバや空調機器の制御を行うことが実際に行われ始めている．運用管理手順のパターンの考え方を活用して，より一般化された形でシステム運用管理とデータセンタの設備管理を統一的に扱うフレームワークを構築することが今後の課題である．その際，現状の個別最適から全体最適を目指すために，全体の整合性をとるためにかえって労力がかかったり，実装が複雑化することのない方法が必要となるなどの課題もある．

第 6 章

結論

6.1 本研究のまとめ

本研究では，大規模化が進むデータセンタにおける情報システム運用の高度化を目的として，運用手順に着目して研究を行った．具体的には，情報システム運用にかかるコストを低減しながら運用対象となる情報システムの信頼性を向上させるための運用方式に関して研究を行った．さらに，近年データセンタにおける電力消費量の増大がコスト面だけでなく，温室効果ガス増加につながるとして社会的問題となっていることから，情報システムの運用の高度化の施策をデータセンタの省電力運用に対しても適用する方法についての研究を行った．

具体的には，第 1 章に述べた下記の課題を解決すべく研究を行った．

課題（1）：情報システムの運用手順は一般的に手順書の形式でまとめられており，ノウハウの一般化ができず，運用の自動化の阻害要因にもなっている．

課題（2）：運用自動化を実現するためには，障害予兆検知，障害発生検知，障害部位特定が重要であるが，そのための有効な情報であるデータセンタ内で膨大に出力されるログ/メッセージ情報の活用が不十分．

課題（3）：情報システム運用とデータセンタの省電力運用は個別に検討されており，改善の余地がある．

課題（1）に対しては，第 2 章で，実際の情報システムの運用手順をモデルに基いて分析し，運用手順のパターンを抽出し，運用手順をパターンに分類して整理することにより，運用自動化のための指針が得られることを示した．また，第 5 章において運用手順のパターンの分類に基いた運用手順の記述方式を提案し，省電力運用手順の記述の例を示すことで，IT 機器だけでなくデータセンタの設備機器を含めた運用手順の記述が可能であることを示した．

課題（2）に対しては，第 3 章及び第 4 章において，ログ/メッセージ情報を活用して「イベント検知」を行う方法について研究を行った．

第 3 章では，データセンタ内で発行される各種メッセージ情報を CEP 技術の一つであるストリーム DB を利用してリアルタイムで分析し，問題発生を検知する手法を提案した．

特にリアルタイム性確保のために，全ての処理をオンメモリで行うため，スケーラビリティ確保のための課題であるメモリ消費量削減のための CQL 設計指針を提案した. この設計指針を確実に遵守し CQL 開発の生産性向上を図るための CQL 自動生成方式を提案した. またストリーム DB 適用の効果を実際に大規模データセンタでの適用実験で実証した.

第 4 章では，キャパシティプラニング時の「設計基準値」と，ログ情報を解析して得られる実際に稼働している時の「実測基準値」の偏差に着目することで，リソース不足あるいはリソース過剰の予兆を検知する方式を提案した.「設計基準値」と「実測基準値」の偏差をストリーム DB を用いて閾値判定，線形外挿で評価する方法に加え，偏差を統計的検定で評価する方式を提案した．いずれの方法についてもプロトタイプによる実験を行い，提案方式の有効性を示すとともに，統計的検定の精度とストリーム DB のメモリ消費量の観点から，最適なスライディングウィンドウのサイズと危険率の設定の指針を与えた.

課題（3）に対しては，第 5 章では，情報システムの運用手順をデータセンタ全体の運用に拡張することで，特に省電力運用を見通しよく行えることを，各パターン毎に省電力運用の例をユースケースとして例示することで示した．また，第 2 章で抽出した運用手順のパターンを基にした運用手順記述方式を提案し，これが省電力効果の算定に利用可能であることを，事例を使って示した.

6.2 今後の課題

課題（1）に関する残された課題としては以下のようなものがある．本研究では，運用手順を最小の基本要素に分解し，それに対してパターンの抽出を行った．今後，複雑な一連の運用手順において，基本要素の運用手順がどのように組み合わされて利用されているかを分析することで，情報システム運用の高度化のための運用手順の設計方法及び，運用自動化に関する知見が得られる可能性があると考える．また，運用手順のパターンに基く運用手順記述方式を，近年注目されている情報システムの運用をワークフローとして定義して自動化を図ろうとする RBA（Run Book Automation）の動きに対応するよう発展させることが必要であると考える.

課題（2）に関しては，第 3 章，第 4 章で取り上げた課題以外にも，ログ/メッセージ情報を活用して問題発生を検知できる課題はさまざまあると考えられる. 本研究の範囲では，個別の障害事象ごとに障害発生時のパターンを分析して，当該パターンを検知する CQL を作成している．将来の研究の方向としては，更に一般化を進め過去のログ/メッセージ情報のデータマイニングにより異常発生時のパターンを自動抽出し，抽出されたパターンを

検知する CQL を自動生成するといった方式をの検討が考えられる．これにより，より広範囲の障害に関して早期に正確に障害発生あるいは障害の予兆を検知可能になると期待できる．

本研究では運用手順の自動化に最も効果があると考えられる「イベント検知」を対象に自動化の検討を行ったが，「アクション」部分については未着手である．動作環境の仮想化技術の進展により「アクション」部分について，環境の差異を意識しないで制御できる範囲が拡大しつつある．このような技術を活用して「アクション」部分に対しての運用の高度化を図っていくことが今後の課題である．

課題（3）については，本研究では情報システムの運用管理とデータセンタの設備機器を統一的に扱う運用手順記述方法を提案したが，実際にこの記述方法に基いて運用手順を自動化するためには，課題（1）のところで述べた RBA への対応及び，「イベント検知」及び「アクション」の自動化対象を拡大していくことが必要である．「イベント検知」及び「アクション」の自動化対象の拡大にあたっては，現状の運用管理手順の分析に基く優先順位付けを行い，それぞれの項目に対して，「イベント検知」であれば本論文の第 3 章，第 4 章で行ったような研究が必要である．

謝辞

本研究の全過程を通じ，懇切なるご指導とご鞭撻を賜りました大阪大学大学院情報科学研究科マルチメディア工学専攻 薦田憲久教授に心から感謝申し上げます.

情報科学研究科博士後期課程において，情報工学全般に関して親切なるご指導とご助言を賜るとともに，本研究をまとめるにあたって貴重なお時間を割いて頂き，丁寧なるご教示を賜りました大阪大学大学院情報科学研究科 マルチメディア工学専攻 藤原融教授，大阪大学サイバーメディアセンター 冨樫祐一講師に謹んで深謝致します.

情報科学研究科博士後期課程において，情報工学全般に関して親切なるご指導とご助言を賜りました，大阪大学大学院情報科学研究科マルチメディア工学専攻 西尾章治郎教授，細田耕教授に深く感謝申し上げます.

最期に頂いた言葉が本研究を始めるきっかけとなる激励の言葉であった故今城哲二博士（元（株）日立製作所ソフトウェア事業部，元東京国際大学教授）に心から感謝しご冥福をお祈りいたします.

博士課程入学のきっかけを与えて頂き，本研究の全般に亘り共同研究者として助言，議論頂いた公立はこだて未来大学情報アーキテクチャ学科教授 大場みち子博士（元（株）日立製作所ソフトウェア事業部）に感謝いたします.

本研究の機会を与えていただくとともに，暖かいご指導とご鞭撻を賜った，（株）日立製作所 ソフトウェア事業部中村孝男前事業部長，坂上秀昭事業部長 尾山壮一事業主管に心から御礼申し上げます．また，研究を進めるにあたり日々様々なご支援とご配慮を頂きました（株）日立製作所 ソフトウェア事業部の先輩，同僚，後輩の方々に心から御礼申し上げます.

第 2 章の研究にあたり，分析の素材となる情報を提供頂いた，（株）日立製作所ソフトウェア事業部先端情報システム開発部の浅見真人担当部長，（株）日立製作所ソフトウェア事業部第 2AP 基盤開発部 相澤宣一主任技師に感謝いたします.

第 3 章の研究に当たり，共同研究者としてご協力頂いた，（株）日立情報システムズシステム運用サービス技術開発部 山出泰子担当部長，（株）日立製作所ソフトウェア事業部第 2AP 基盤開発部 中道繁技師に感謝いたします．また，実際のデータセンタの運用についてご教示頂きデータセンタでの適用実験に多大なご協力を頂いた，（株）日立情報システムズアウトソーシング事業部 塩谷隆廣本部長，和田善也本部長，川崎敦主任技師に御礼申し

上げます.

第 4 章の研究に当たり，共同研究者としてご協力頂いた,（株）日立製作所ソフトウェア事業部第 2 基盤ソフト設計部　半田敦郎企画員に感謝いたします.

第 3 章および第 4 章の研究を推進するにあたり，ストリーム DB エンジン並びにマニュアル等の技術資料の提供および実験の実施に協力頂いた,（株）日立製作所ソフトウェア事業部第 2 基盤ソフト設計部 五百木伸洋担当部長をはじめとする uCosminexus Stream Data Platform 開発担当チームの皆様に感謝いたします.

第 5 章の研究にあたり，共同研究者としてご協力頂いた,（株）日立製作所ソフトウェア事業部 JP1 販売推進グループ 西部憲和主任技師に感謝いたします．また，省電力効果の検証にご協力頂いた,（株）日立製作所ソフトウェア事業部企画本部 和田栄 本部長付，同本部共通企画部 齋藤寿彦主任技師に御礼申し上げます．さらに，コラボレーションツールの運用状況についてご教示頂きました,（株）日立製作所情報システム事業部 飯田透主任技師, ならびにコラボレーションツールのエンハンス内容についてご教示頂きました,（株）日立製作所ソフトウェア事業部 第 3 基盤ソフト設計部 永山光春部長に御礼申し上げます.

最後に，研究を支えてくれた家族に感謝します，特に本論文執筆中に亡くなった父親に感謝するとともに，冥福を祈ります.

参考文献

[1] U.K. Office of Government Commerce: "*Applications Management: ITIL(Information Technology Infrastructure Library)*," Stationery Office, London, United Kingdom, 2002.

[2] P. Naur and B. Randell (Eds.): "Software Engineering," *Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*, Scientific Affairs Division, NATO, 1969.

[3] B. Randell and J.N. Buxton (Eds.): "Software Engineering Techniques," *Report of a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969*, Scientific Affairs Division, NATO, 1970.

[4] 経済産業省："平成 18 年度情報処理実態調査報告書," http://www.meti.go.jp/statistics/zyo/zyouhou/result-2/pdf/3_H18report_rev071204.pdf, 2007.

[5] M. Miller: "*Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*," Que, 2008.

[6] M. Turner, D. Budgen, and P. Brereton: "Turning Software into a Service," *IEEE Computer*, Vol. 36, No. 10, pp.38-44, 2003.

[7] L. A. Barroso and U. Holzie: "*The Data Center as a Computer: An Introduction to the Design of Warehouse-Scale Machines,"* Morgan & Claypool, 2009.

[8] A. G. Ganek and T. A. Corbi: "The Dawning of the Autonomic Computing Era," *IBM System Journal*, Vol. 42, Issue 1, pp.5-18, 2003.

[9] G. Lanfranchi, P.D. Peruta, A. Perrone, and D. Calvanese: "Toward a New Landscape of Systems Management in an Autonomic Computing Environment," *IBM System Journal*, Vol. 42, Issue 1, pp.119-128, 2003.

[10] J.O. Kephart and D.M. Chess: "The Vision of Autonomic Computing," *IEEE Computer*, Vol.36, Issue 1, pp.41-50, 2003.

[11] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland: "A Concise Introduction to Autonomic Computing," *Advanced Engineering Informatics*, Vol. 19 , Issue 3, 2005.

[12] 吉野松樹，阿部欣成，中誠一郎："ビジネスグリッドの狙い，" 情報処理，Vol.47，No.9, pp.947-952, 2006 .

[13] 宮川伸也，佐治信之，工藤裕，田崎英明："ビジネスグリッド技術解説，" 情報処理，Vol.47, No.9, pp.953-961, 2006.

[14] 幕田幸男，佐々木一陽，阿部秀哉，藤野修司: “ビジネスグリッドの実証実験,” 情報処理, Vol.47, No.9, pp.962-969, 2006.

[15] 福井恵右，岸本光弘，佐川暢俊，中田登志之，森拓也，舘村純一: “ビジネスグリッド関連技術動向と標準化活動, ” 情報処理, Vol.47, No.9, pp.970-977, 2006.

[16] 独立行政法人 情報処理推進機構: “ビジネスグリッドが切り開く次世代 IT 基盤, ” アスキー，2006.

[17] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, “Fulfilling the Vision of Autonomic Computing, ” *IEEE Computer*, Vol. 43, Issue 1, pp.35-41, 2010.

[18] T. Mudge: “Power: A First-class Architectural Design Constraint,” *IEEE Computer*, Vol. 34, Issue 4, pp.52-58, 2001.

[19] D. Colarelli and D. Grunwald: “Massive Arrays of Idle Disks for Storage Archives,” in *Proc. of the 2002 ACM/IEEE Conference on Supercomputing*, pp.1-11, 2002.

[20] 朝康博，中島忠克，沖津潤，加藤猛，齊藤達也，頭島康博: “環境配慮型データセンタ向けIT 連係空調最適化制御方式,” 第9回情報科学技術フォーラム予稿集　第1分冊, pp.97-102(RC-009), 2010.

[21] IBM: “*An Architectural Blueprint for Autonomic Computing*, ” http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf, 2006.

[22] G. Kaiser, J. Parekh, P. Gross, and G. Valetto: “Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems,” in *Proc. of the Autonomic Computing Workshop 5th Annual International Workshop on Active Middleware Services(AMS’03)*, pp.20-30, 2003.

[23] A.S.M. De Franceschi, K.S. Borges, R. Moraes, and F. Vasques : “Autonomic Computing Systems: Using AI Techniques for the Development of Agents in the Network Management Domain,” *WSEAS Transaction on Communications*, Vol.5, Issue 8, pp.1353-1360, 2006.

[24] 佐藤善郎，更田洋吾，長田充弘，増岡義政，谷村武洋: “ポリシーベース自律運用を実現する運用管理ソリューション,” 日立評論，Vol.87, No.4，pp.45-50, 2005.

[25] A. Melamed: “Distributed Systems Management on Wall Street – AI Technology Needs,” in *Proc. of the 1st International Conference on Artificial Intelligence on Wall Street*, pp.206-212, 1991.

[26] J. O. Kephart and W. E. Walsh: “An Artificial Intelligence Perspective on Autonomic Computing Policies,” in *Proc. of 5th IEEE International Workshop on Policies for Distributed*

Systems and Networks(POLICY 2004), pp.3-12, 2004.

[27] J.L. Hellerstein: "*Automating Performance Management Using Case-Based Reasoning*," Technical Report RC-20083, IBM TJ Watson Research Center, 1995.

[28] T. Koch and B. Kramer: "Rules and Agents for Automated Management of Distributed Systems," *Distributed Systems Engineering Journal*, Vol.2, Special issue on Distributed Systems Management, pp.110-114, 1996.

[29] 山出泰子: "メッセージ分析システム及びメッセージ分析プログラム," 特許公開 2006-331026, 2006.

[30] 後藤邦仁，望月秀樹，宗像勉，泉全: "IT サービス運用を最適化するソリューション," 日立評論, Vol.87, No.4, pp. 359-362, 2006.

[31] J. Allspaw : "*The Art of Capacity Planning: Scaling Web Resources*," O'Reilly Media, 2008.

[32] D.A. Menasce, L.W. Dowdy, and V.A.F. Almeida: "*Performance by Design: Computer Capacity Planning By Example*," Prentice Hall, 2004.

[33] M. Pinzger: "Automated web performance analysis, with a special focus on prediction," in *Proc. of the 10th International Conference on Information Integration and Web-based Applications & Services*, pp.539-542, 2008.

[34] S. Lin, X. Guo, W. Chen, Z. Zhang, and Y. Lin, "An Automation Model for the Building Energy Management Systems. A Theoretical Study," in *Proc. of the 10th WSEAS International Conference on Robotics, Control and Manufacturing Technology (ROCOM'10)*, pp.41-46, 2010.

[35] 沖津潤，加藤猛，齊藤達也，平島陽子，中島忠克，頭島康博: "次世代データセンタ向け環境配慮運用管理方式（1）," 電子情報通信学会大会論文講演集，Vol.2009, 通信ソサイエティ 2, p.335 （B-14-4）, 2009.

[36] 平島陽子，沖津潤，加藤猛，齊藤達也，中島忠克，頭島康博: "次世代データセンタ向け環境配慮運用管理方式（2）," 電子情報通信学会大会論文講演集，Vol.2009, 通信ソサイエティ 2, p.336 （B-14-5）, 2009.

[37] 中島忠克，沖津潤，加藤猛，齊藤達也，平島陽子，頭島康博: "次世代データセンタ向け環境配慮運用管理方式（3）," 電子情報通信学会大会論文講演集，Vol.2009, 通信ソサイエティ 2, p.337 （B-14-6）, 2009.

[38] 荒井大輔, 吉原貴仁: "空調機の電力消費量を考慮したサーバ仮想化とネットワーク動的構成変更によるデータセンタ省電力運用管理手順の提案," 第8回情報科学技術フォーラム予稿集　第4分冊, pp.183-184（L-023）, 2009.

[39] M. Yoshino, N. Komoda, and M. Oba: “An Analysis of Patterns for Automating Information System Operations,” *WSEAS Transactions on Information Science & Applications*, Vol. 5, Issue 11, pp.1618-1627, 2008.

[40] M. Yoshino, N. Komoda, and M. Oba: “Creating Operation Method Patterns for Automating Information System Operations,” in *Proc. of the 8th WSEAS International Conference on Applied Computer Science (ACS'08)*, pp.171-175, 2008.

[41] 吉野松樹，薦田憲久，大場みち子: “運用自動化に向けた運用手順のパターン化,” 電気学会情報システム研究会, IS-08-41, pp.35-38, 2008.

[42] 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Ｃ部門論文誌，Vol.130，No.4, pp.607-614, 2010.

[43] M. Yoshino, M. Oba, N. Komoda, T. Yamade, and S. Nakamichi: “Message Analysis Method Based on a Stream Database for Information System Management,” in *Proc. of the 4th International Conference on Research Challenges in Information Science 2010 (RCIS2010)*, pp.519-526, 2010.

[44] 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,” 電気学会情報システム研究会, IS-09-62, pp.37-42, 2009.

[45] 吉野松樹，薦田憲久，半田敦郎，大場みち子: “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,” 第 9 回情報科学技術フォーラム（FIT2010）予稿集 第 4 分冊, pp.123-128（RO-001）, 2010.

[46] 吉野松樹，薦田憲久，半田敦郎，大場みち子: “ストリーム DB を用いた Web システム稼動監視方式,” 電気学会Ｃ部門論文誌（投稿中）.

[47] M. Yoshino, N. Komoda, and M. Oba: “On a Method for Describing System Management Operations and its Use in Evaluating Energy Saving Operations,” in *Proc. of the 9th WSEAS International Conference on Data Networks, Communications, Computers (DNCOCO '10)*, pp.47-52, 2010.

[48] M. Yoshino, N. Komoda, N. Nishibe, and M. Oba: “Classification of Energy-Saving Operations from the Perspective of System Management,” in *Proc. of 8th IEEE International Conference on Industrial Informatics 2010 (INDIN2010)*, pp.651-656. 2010.

[49] 吉野松樹，薦田憲久，大場みち子，西部憲和: “情報システム省電力運用のシステム運用観点での分類,” 電気学会情報システム研究会, IS-01-26, pp.135-140, 2010.

[50] M. Yoshino, M. Oba, and N. Komoda: “Extending a Method of Describing System Management Operations to Energy-Saving Operations in Data Centers,” *Internatinal Journal of Computers*, Vol. 5, Issue 1, pp.115-122, 2011.

[51] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides: “*Design Patterns: Elements of Reusable Object-Oriented Software*,” Addison-Wesley, 1994.

[52] T.J. Mowbray and R.C. Malveau: “*CORBA Design Patterns*,” John Wiley & Sons , 1997.

[53] D. Adamopoulos: “Design Patterns in Telecommunications Service Engineering,” in *Proc. of 8th International Conference on Information Integration and Web-based Application & Services(iiWAS2006)*, pp. 249-256, 2006.

[54] M. Fowler: “*Patterns of Enterprise Application Architecture*,” Addison-Wesley Professional, 2002.

[55] 吉野松樹: “日立製作所の「Harmonious Computing」を中心としたビジネスグリッドコンピューティングへの取り組み（特集 1 ビジネスグリッドコンピューティング）,” COMPUTER & NETWORK LAN, オーム社, Vol. 22, No.8 （通巻 250 号）, pp.54-62, 2004.

[56] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aals, and N. Mulyar : “Workflow control-flow patterns a revised view,” http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf.

[57] D. Luckham: “*The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*,” Addison-Wesley Professional, 2002.

[58] 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.

[59] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk: “Gigascope: A stream database for network applications,” in *Proc. of the 2003 ACM SIGMOD International Conference. on Management of Data*, pp.647-651, 2003.

[60] S. Chakravarthy and Q. Jiang: “*Stream Data Processing: A Quality of Service Perspective: Modeling, Scheduling, Load Shedding, and Complex Event Processing*,” Springer, 2009.

[61] 樫山俊彦, 花井知広, 田中美智子, 今木常之, 西澤格: “ストリームデータ処理におけるデータベースアーカイブ処理高速化の提案と評価,” 日本データベース学会論文誌, Vol. 6, No. 2, pp.37–40, 2007.

[62] 今木常之, 西澤格: “ストリームデータ処理におけるデータ生存期間管理方式,” 日本データベース学会論文誌, Vol.5, No.2, pp.65-68, 2006.

[63] 今木常之，樫山俊彦，西澤格: “遅延演算を利用したストリームデータの再帰処理方法, ” 日本データベース学会論文誌， Vol.8, No.4, pp.7-12, 2010.

[64] 川島英之， 寺島裕貴, 北川博之: “ストリーム処理エンジンにおける効率的な来歴管理, ” 日本データベース学会論文誌, Vol.8, No.1, pp.101-106, 2010.

[65] 森有一， 角谷有司， 西澤格， 馬場恒彦: “情報爆発時代の到来に向けた大量高速データ処理技術への取り組み,”日立評論, Vol.90, No.7, pp. 612-615, 2008.

[66] Oracle Corporation: “Oracle CEP CQL Language Reference, 11*g* Release 1 (11.1.1) ,” E12048-01, http://download.oracle.com/docs/cd/E12839_01/doc.1111/e12048.pdf.

[67] P. Mishra and M.H. Eich: “Join Processing in Relational Databases,” *ACM Computing Surveys*, Vol.24, No.1, pp.63-113, 1992.

[68] 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.

[69] P. Pande, R. Neuman, and R. Cavanagh: “*The Six Sigma Way: How GE, Motorola, and Other Top Companies are Honing Their Performance*,” McGraw-Hill, 2000.

[70] 伊藤清: “確率論,” 岩波書店, 1991.

[71] 杉原左右一: “統計学　増補第 2 版,” 晃洋書房, 2003.

[72] 小堀一雄，茂呂範，佐藤聖規，石垣一，飯山教史: “現場で使えるデバッグ&トラブルシュート（Java 編），” 翔泳社, 2010.

[73] 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.

[74] K Kato, M. Nishimura, and J. Katsumi: “Data Center Operation for Cloud Computing Age Implemented with MasterScope Ver. 8,” *NEC Technical Journal*, Vol.5, No.2, pp.85-90, 2010.

[75] A. Gore: “*Inconvenient Truth: The Planetary Emergency of Global Warming and What We Can Do About It,”* Rodale Pr., 2006.

[76] S. Murugesan: “Harnessing Green IT: Principles and Practices,” *IT Professional*, Vol.10, No.1, pp.24-33, 2008.

[77] J. Lamb: “*Greening of IT, The: How Companies Can Make a Difference for the Environment*,” IBM Press, 2009.

[78] L.A. Barroso: "The Price of Performance: An Economic Case for Chip Multiprocessing," *ACM Queue*, Vol. 3, Issue 7, pp.48-53, 2005.

[79] G. Spafford, *"The Governance of Green IT: The Role of Processes in Reducing Data Center Energy Requirements,"* IT Governance Ltd., 2008.

[80] B. Tomlinson: "*Greening through IT: Information Technology for Environmental Sustainability*," MIT Press, 2010.

[81] M. Amin and B.F. Wollenberg: "Toward a smart grid: Power delivery for the 21st Century, " *IEEE Power and Energy Magazine*, Vol. 3, no. 5, pp.34-41, 2005.

[82] R. Bianchini and R. Rajamony: "Power and Energy Management for Server Systems, " *IEEE Computer*, Vol. 37, No. 11, pp.68-74, 2004.

[83] DMTF(Distributed Management Task Force), *DSP0226 WS-Management(Web Services for Management)*, http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf, 2008.

[84] W. Gentzsch, K. Iwano, D. Johnston-Watt, M.A. Minhas, and M. Yousi: "Self-adaptable Autonomic Computing Systems: An Industry View," in *Proc. of 16th International Workshop on Database and Expert Systems Applications*, pp.201–205, 2005.

[85] I. Michael: "Autopilot: Automatic Data Center Management," *ACM SIGOPS Operating Systems Review*, pp.60-67, 2007.

[86] E.A. Brewer: "Lessons from Giant-Scale Services," *IEEE Internet Computing*, Vol. 5, No. 4, pp. 46-55, 2001.

[87] M. Chetty, D. Tran, and R.E. Grinter: "Getting to Green: Understanding Resource Consumption in the Home," in *Proc. of the 10th International Conference on Ubiquitous Computing*, pp. 242-251, 2008.

[88] T.D. Boucher, D.M. Auslander, C.E. Bash, C.C. Federspiel, and C.D. Patel: "Viability of Dynamic Cooling Control in a Data Center Environment," *Journal of Electronic Packaging*, Vol. 128, Issue 2, pp.137-144, 2006.

[89] M. Rosenblum and T. Garfinkel: "Virtual Machine Monitors: Current Technology and Future Trends," *Computer*, Vol. 38, No. 5, pp.39-47, 2005.

[90] J. Sanekata, M. Tani, T. Nishii, Y. Hamatsuka, and H. Sugai: "Environmental Impact Assessment Logic for ICT systems named SI-LCA and its Case Example," in *Proc. of the 2007 IEEE International Symposium on Electronics and Environment*, Vol. 38, No. 5, pp.6-11, 2007.