情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　221–234 (Sep. 2010)

ウェブアプリケーションの性能異常兆候検出への管理図の適用

岩田　聡[†1]　河野健二[†1,†2]

ウェブアプリケーションの性能異常が重要な問題となりつつあり，性能異常の発生をいち早く検出し，被害が致命的になるのを防ぐことが求められている．しかし，性能指標の 1 つとして用いられるリクエストの処理時間は，正常時でも揺らぎが大きいため，その微妙な変化から性能異常の兆候を検出することは難しい．そこで，本論文では管理図という統計的手法を利用して，リクエストの処理時間に表れる微妙な変化から性能異常兆候の検出を試みる．その際，検出の精度向上や検出後の原因究明に有益な情報を得るために 2 つの工夫を行っている．1 つは，監視対象の値として個々のリクエストの処理時間ではなく，一定時間ごとのリクエストの処理時間に関する 4 種類の統計値を用いている点である．もう 1 つはウェブアプリケーション全体ではなく，リクエストの種類ごとに管理図を作成する点である．実際に RUBiS というウェブアプリケーションに管理図を適用した結果，性能異常の兆候をつかむことができた．管理図によって検出されたいくつかの性能異常の兆候を詳細に調査した結果，(1) データベースへのインデックスの追加および，(2) 性能パラメータの調整などを行うことによって，検出された性能異常が解決できることを確認した．

Applying Control Charts to Detection of Performance Anomaly Symptoms in Web Applications

Satoshi Iwata[†1] and Kenji Kono[†1,†2]

Performance anomaly is becoming a serious problem in web applications. To prevent performance anomaly, it is useful to detect a symptom of performance anomaly to proactively take action against it. Unfortunately, a symptom of performance anomaly is a slight change in processing time in a web application. Thus it is difficult to detect the symptoms without being confused by natural fluctuations in processing time. In this paper, we apply *control charts*, a statistical method to detect deviations from the standard quality of products, to detecting symptoms of performance anomaly. In applying control charts, we devise some means to improve detection accuracy and gain useful information for debugging performance anomaly. To demonstrate the usefulness of control charts, we conducted case studies with *RUBiS*, an auction site modeled after ebay.com. Control charts detected some symptoms of performance anomaly: (1) the increase in processing time due to inappropriate design of database index, and (2) the increase in processing time due to improper setting of performance parameters. We confirmed that the detected anomalies can be fixed by modifying database design and performance parameters.

1. はじめに

ウェブアプリケーションの性能異常が重要な問題となりつつある．性能異常とは，運用時の性能が設計時に想定していた性能より著しく劣化することであり，たとえば，リクエストの応答時間の増大やサーバのスループットの低下などがある．性能異常が起こると Service Level Agreement（SLA）を保証できずに損害賠償が発生したり，サービスの応答性劣化によりユーザ獲得の機会を失ってしまったりすることもある．しかし，性能異常の原因は多岐にわたり，その発生を未然に防ぐのは困難である．ワークロードや資源の利用率，サーバの稼働時間などさまざまな要因によって性能異常は発生する．たとえば，ワークロードの変化にともない，現在のサーバ設定が不適切になり性能異常が発生することがある．また，たとえば，サーバの稼働時間が長くなるにつれ，アプリケーションのバグによるメモリリークが顕在化し，性能異常が発生することもある．

そこで，性能異常の発生をいち早く検出し，性能異常による被害が致命的になるのを防ぐことが求められている．性能異常の兆候を検出できれば，性能異常がユーザに気づかれるほど致命的になる前に対策を行える可能性が増す．仮に異常の兆候を検出した後迅速に原因究明と修復を行うことができれば，性能異常がユーザに気づかれることなく，システムを回復することができる．また原因究明を即座に行うことができなかった場合でも応急処置として，サーバの追加を行い性能異常がユーザに気づかれるのを回避したり，サービスにアクセス可能なクライアント数を一時的に制限することでより深刻な問題が発生するのを回避したりすることができる．

†1 慶應義塾大学
Keio University
†2 科学技術振興機構戦略的創造研究推進事業
Japan Science and Technology Agency（JST）/CREST

しかし現在，性能異常兆候の検出は人手を介したアドホックな手法が主流になっている．たとえば，リクエスト処理時間の平均値が事前に定めた閾値を超えたことをもって性能異常兆候と検出したり，直近の平均値が増加傾向にあることをもって性能異常兆候と検出したりする．このような手法では閾値を何秒にするのか，また増加傾向が何分継続したことをもって性能異常兆候だと検出するかを適切に定めるのが難しく，経験則に頼っているのが実状である．

そこで，本論文では管理図という統計的手法を利用して，ウェブアプリケーションの性能異常の兆候検出を試みる．管理図は元々，製品製造工程や経営工程を管理するためのものであり，工程が統計的に正常か異常かを判定するものである[1)]．過去の管理対象の値（例：ネジの長さ，企業利益，コレステロール値）を用いて基準線を計算しておき，現在の管理対象の値と基準線とを比較することで管理対象の値に統計的な変化が見られるかどうかを判断する．そして，管理図が管理対象の値に統計的に変化が生じたと判断し警告を発した場合は，工程に変化が生じた（例：工具の摩耗，取引先の減少，食生活の変化）はずであると考え原因究明を始める．管理図は過去の品質と現在の品質のずれを統計的に判定するものであるため，管理図を用いてウェブアプリケーションの過去の処理時間と現在の処理時間を比較することで，処理時間に表れる性能異常の兆候を経験則に頼らずシステマティックに検出できると期待できる．本論文でリクエスト処理時間とはサーバがリクエストを受信してから処理結果を送信するまでの時間を指し，人間のユーザとインタラクティブに処理を行うウェブアプリケーションでは，このリクエスト処理時間が重要な性能指標となる．本論文でも性能指標としてリクエスト処理時間を用いる．現在，侵入検知のために管理図を利用した研究は存在する[2)]が，性能異常の兆候を検出するために管理図をウェブアプリケーションへ適用した研究は著者らの知る限りでは存在しない．

管理図は以下にあげるような利点があり，ウェブアプリケーションの性能異常の兆候を検出するのに適している．管理図は人間が決定しなければならないパラメータが少なく，計算量も少ないため導入が容易である．精度に大きく影響を及ぼすパラメータを人間が決定する必要がある場合，その利用者が統計的意味を正確に理解してから手法を適用する必要があるため導入が困難になる．また，計算量が多い手法では，たとえば実運用時に利用できないなど，適用範囲が限られてしまう．さらに管理図は，管理対象の値に特別な制約がないため，ウェブアプリケーションの特性や管理者の要求に応じてさまざまな適用が可能だと考えられる．管理図にはさまざまな種類があり，管理対象の値に応じて適切な管理図を選ぶことができる．さらに，管理図により得られる結果は「正常」であるか「異常」であるかの 2 通りであり，管理者にとって理解しやすい形式である．

本論文では，管理図をウェブアプリケーションの性能異常兆候の検出に適用するために，2 つの工夫を行っている．1 つは，管理対象の値として個々のリクエストの処理時間ではなく，一定時間ごとのリクエストの処理時間に関する 4 種類の統計値（平均値，最大値，中央値，最小値）を用いている点である．このようにすることで処理時間の揺らぎに対して堅牢なものとなっている．もう 1 つはウェブアプリケーション全体ではなく，リクエストの種類ごとに管理図を作成する点である．リクエストの種類ごとに管理図を作成することで，異常兆候検出後の原因究明に有益な情報を得ることができるようにもなる．

本論文では，オークションサイト eBay.com[3)] を模したベンチマーク用ウェブアプリケーションである RUBiS[4)] の性能異常の兆候検出に管理図を応用した結果を報告する．管理図を用いることで実際に RUBiS の性能異常の兆候を 4 つ検出することができた．これらの性能異常は，(1) サービス運用の経過にともなう，データベース内のテーブルへのインデックス不足によるリクエストの処理時間の増大と，(2) クライアント数の増加にともない性能パラメータの設定が不適切になってしまったことによるリクエストの処理時間の増大に大別できる．(1) の性能異常はデータベース内のテーブルへインデックスを追加し，(2) の性能異常はウェブサーバ Apache の `KeepAlive` に関するパラメータとアプリケーションサーバ JBoss の `maxThreads` を適切に設定し直すことで性能を回復できた．

この実験を通じて，管理図が発した警告から原因究明を行う際に注意すべき 3 つの知見を得た．1 つめは，より重要な性能異常の原因特定を行うために，長期間にわたって管理図が警告を発しているリクエストの種類に注目すべきだということである．2 つめは，性能異常は伝播していくことがあるため，性能異常の発生源を特定するには一番最初に管理図が警告を発したリクエストの種類に注目すべきだということである．3 つめは，性能異常の原因となっているコンポーネントをあるリクエストの種類が利用していたとしても，必ずそのリクエストの種類の管理図が警告を発するとは限らないということである．

以下，2 章で管理図の説明を行う．3 章では，ウェブアプリケーションの性能異常の兆候を検出するために，本論文で行った管理図の適用方法を説明する．そして 4 章で管理図を RUBiS に適用して性能異常の兆候を検出できたケーススタディを報告する．その後 5 章で，実験により得た，原因究明を行う際の注意事項を述べる．さらに 6 章で関連研究について述べ，最後に 7 章で本論文をまとめる．

2. 管　理　図

本章では管理図の説明を行う．管理図をすでに知っている読者は本章を読み飛ばし，3 章へすすんでもかまわない．

管理図は元々，製品製造工程や経営工程を管理するためのものであり，工程が統計的に正常か異常かを判定するものである．管理図は過去の品質と現在の品質のずれを統計的に比較し，現在の工程が正常か異常かの判定を行う．

管理図では管理対象の値を**特性値**と呼び，その特性値を監視する．そして，その特性値が統計的に異常な値となった場合に管理図は警告を発する．工場でのネジ製造工程を管理する場合を例にとって説明する．このネジ製造工程では特性値をネジの長さと定め管理する．平常時においては，ネジの長さの誤差はある一定の値にとどまり，管理図は警告を発しない．逆に，もし管理図が警告を発した場合にはネジ製造工程に何かしらの変化が生じたと考えることができ，工具や材料の検査など原因究明を始める必要がある．このように，管理図を用いることで，ネジが不良品になってしまうほどネジの長さの誤差が大きくなってしまう前に異常を検出することができる．

管理図には多くの種類があり，特性値の性質と活用目的により適切なものを選択することが必要である．まず，特性値が計量値か計数値かといったように特性値の性質によって利用する管理図の種類は異なってくる．さらに，特性値が計量値の場合でも，異常を早期に検出したいのか，それとも検出の精度を高くしたいのかといったように活用目的によって利用する管理図の種類は異なってくる．このように，管理図は，ウェブアプリケーションの特性や管理者の要求に応じてさまざまな形で性能異常兆候の検出に適用可能だと考えられる．

図 1 が管理図の例である．横軸に日や時刻やロット番号などをとり，縦軸に特性値の統計値（例：平均値，中央値，不良率）をとり，打点していく．図 1 は特に上で述べた，工場

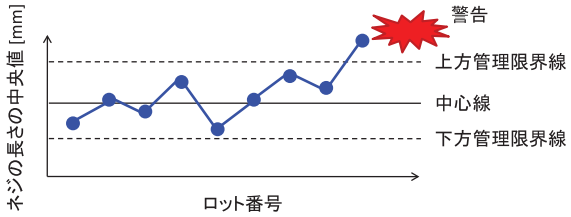


図 1　管理図の例
Fig. 1　Example of a control chart.

でのネジ製造工程を管理する場合の例であり，ロットごとにネジの長さ（特性値）の中央値（統計値）を計算し打点していく．その際，工程が正常か異常かを判定するために，図 1 にあるとおりあらかじめ過去に測定した特性値のデータから**中心線**，**上方管理限界線**，**下方管理限界線**の 3 本の基準線を引いておく．

本論文では**メディアン管理図**を用いるので，ここではメディアン管理図についてのみ作成手順を説明する．本論文でメディアン管理図を選択した理由は 3 章で述べる．説明では上で述べた，工場でのネジ製造工程を管理する場合の例も付随させる．また，管理図の数理を付録にしたので，興味のある読者はそちらを参照しながら読み進めてほしい．

まず初めに基準線を決定するために正常時に 100 個の特性値を測定し，その特性値を測定した順番に 5 個ずつ 20 の群に分類する．ネジ製造工程の例では，ロット番号順に並べた 20 ロットから各 5 個ずつネジを選択し，その長さを測定する．ここで，特性値は 100 個測定するのが慣例となっている．また，群の大きさは付録で述べてあるとおり 4 以上を選択する必要があり，また中央値の計算を容易にするために通常は奇数である 5 を選ぶ．本論文で正常時をいつと定めるかについては 3.3 節で述べる．そして，各群について中央値 Me と範囲 R（最大値と最小値の差）を計算する．この 20 個の中央値 Me の平均値 $\overline{Me}$ が中心線となる．さらに 20 個の範囲 R の平均値 $\overline{R}$ を計算し，2 本の管理限界線を求める式

$$UCL,\ LCL = \overline{Me} \pm A_2\overline{R}$$

に代入する．UCL が上方管理限界線（Upper Control Limit），LCL が下方管理限界線（Lower Control Limit）である．ここで A_2 は群の大きさによって一意に定まる値であり，群の大きさが 5 のとき A_2 は 0.69 になる．この 2 本の管理限界線を求める式の導出は付録を参照してほしい．

そして，工程を管理する際には特性値を 5 個測定するごとにその中央値を計算し，管理図の上に打点していく．ネジ製造工程の例では，ネジが 1 ロット完成するごとに 5 個ずつネジの長さを測定し，中央値を計算し，管理図に打点していく．ここで，特性値そのものではなく，特性値の統計値（メディアン管理図では中央値）を打点していくことに注意してほしい．

打点した特性値の統計値が管理限界線の内側にプロットされ，かつ点の並びにクセがなければ工程は統計的に正常であると判断される．逆に，図 1 のように管理限界線を超えて打点されるか，または管理限界線の内側でも打点の並びにクセがあると，工程は異常であるという警告になる．打点の並びのクセとは，中心線とどちらか一方の管理限界線の間に 9 点の連があったり，6 点が増加または減少傾向で並ぶことなどである．ただし，本論文のケーススタディで観測した警告はすべて管理限界線を超える打点により発せられたものであり，

管理限界線の内側での打点の並びのクセによる警告は観測していない．しかし，管理限界線を超えずに少しずつリクエストの処理時間が増大し続けるような性能異常も発生する可能性はあると考えている．本論文ではこのクセを判定するルールの数理については詳細に説明せず，説明は参考文献 1)，5) などにまかせる．

ここで，ウェブアプリケーションのリクエストの処理時間を特性値とした場合に，下方管理限界線を超える打点を性能異常兆候だと検出することについて議論する．通常，リクエストの処理時間は小さいほど性能は良いと判断される．しかし，異常に処理時間が小さい場合，性能異常の兆候を示している可能性がある．それは，通常行わなければならない処理を行わなかったために，処理時間が小さくなっている可能性があるからである．たとえば，本来解放されるべきキャッシュの解放が適切に行われておらず，参照すべきではないキャッシュを参照してしまった場合がこれにあたる．このため，本論文では通常の管理図同様，下方管理限界線を超える打点も警告と判断する．

3. 性能異常兆候検出への管理図の適用

本論文ではメディアン管理図を用いる．その理由は，メディアン管理図が揺らぎに強いからである．ウェブアプリケーションの処理時間は平常時でも揺らぐことがある．原因はさまざま考えられるが，たとえばスケジューリングやロックの挙動に影響を受けている可能性がある．**図 2** は 10 分間ごとにあるリクエストの種類の処理時間の平均値を計算したグラフである．図 2 を見ると，平均値は約 30 ミリ秒から約 120 ミリ秒と大きく揺らいでいることが分かる．ずれに敏感な管理図を用いた場合，これらの揺らぎまで異常と検出してしまい，

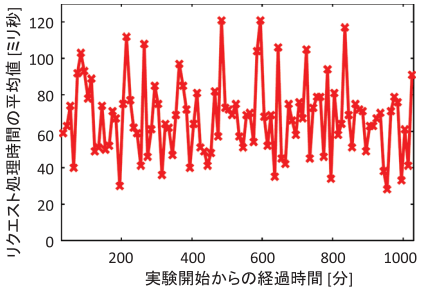


図 2　リクエスト処理時間の平均値．オークションサイト RUBiS のトップページを取得するリクエストの処理時間の平均値を打点した．平均値の計算は 10 分ごとに行った
Fig. 2　Average of request processing time.

警告が多発してしまう可能性がある．そこで，個々の管理図の特性をふまえ，測定値の揺らぎに強いメディアン管理図を選択した．

本論文では，管理図をウェブアプリケーションの性能異常兆候の検出に適用するために，さらに 2 つの工夫を行っている．それらの工夫を行い，本論文で作成する管理図は**図 3** のようになる．1 つめの工夫は特性値として個々のリクエストの処理時間ではなく，一定時間ごとにリクエストの処理時間の統計値を 4 種類（平均値，最大値，中央値，最小値）計算し，それらを選択するということである．本論文ではメディアン管理図を用いるので，つまり，4 つの管理図を用意し，平均値の中央値，最大値の中央値，中央値の中央値，最小値の中央値を打点していくことになる．個々のリクエストの処理時間は揺らぎが大きいため，ある程度まとまった数のリクエストの処理時間の統計値を管理対象とする方が処理時間の揺らぎに対して堅牢である．また，この工夫を行うことでリクエストの処理時間にさまざまな形で表れる性能異常を検出することができる．さらに，この工夫により，ウェブアプリケーションの管理者が個々のウェブアプリケーションの性質に応じてリクエストの処理時間を管理できる．2 つめの工夫は管理図をウェブアプリケーション全体で 1 つではなく，リクエストの種類ごとに作成するということである．こうすることで，異常の兆候をより早期に検出する可能性が高くなり，さらに，異常兆候検出後の原因究明に有益な情報を得ることができるようにもなる．RUBiS の例では統計値が 4 種類でリクエストが 27 種類のため，管理図は 4×27 で 108 作成することになる．

3.1 管理図の特性値

本論文では管理する特性値として一定時間ごとに計算したリクエストの処理時間の統計値を用いる．一定時間とはたとえば 10 分といった値であり，統計値としては平均値，最大

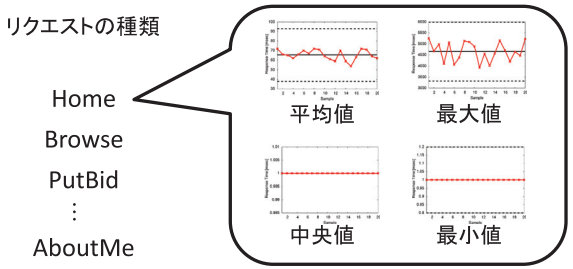


図 3　本論文で作成する管理図
Fig. 3　Control charts used in this paper.

値，中央値，最小値の 4 種類を用いる．

リクエストの処理時間の測定はシステムへの変更が必要なく，またはもし必要だった場合でも少ない変更で可能なため，実システムで行うことはそれほど困難ではないと考えている．たとえば Apache ウェブサーバは個々のリクエストの要求した URL とそのリクエストの処理にかかった時間をマイクロ秒単位でログに記録する機能を備えており，設定を行うだけで個々のリクエストの処理時間を測定できる．

特性値として個々のリクエストの処理時間ではなく，一定時間ごとのリクエストの処理時間の 4 種類の統計値を採用する理由は 3 つあり，それぞれについて説明する．1 つめに，個々のリクエストの処理時間の揺らぎが大きいため，ある程度まとまった数のリクエストの処理時間の統計値を管理対象とする方が処理時間の揺らぎに対して堅牢である．

2 つめに，性能異常の兆候はその原因によってさまざまな形でリクエストの処理時間に表れる．そのため，一般的な性能指標である平均値だけではなく，数種類の統計値を特性値とすることで，小さな変化を見逃さずに性能異常の兆候を検出する．たとえば，同時に処理できる上限数を超える数のリクエストがサーバに到達した場合，一部のリクエストは待ち行列に入れられる．このとき，待ち行列に入れられなかったリクエストの処理は平常時と同様に行われるため，処理時間に変化はない．しかし，待ち行列に入れられたリクエストのみ異常に処理時間が大きくなる．このような場合，性能異常は処理時間の最大値に顕著に表れる．また，中央値を特性値として用いることにより，多数のリクエストの処理時間がわずかずつ増大もしくは減小したことを検出することができる．平均値のみを特性値としていたのでは，少数の，処理時間が異常に大きなリクエストの影響で，このわずかな変化が隠蔽されてしまう可能性がある．さらに，最小値を特性値として用いることにより，リクエストの処理内容が変化したことによる性能異常の兆候を検出することができるようになる．最小値は通常それほど揺らがない．たとえば，資源不足により処理が滞った場合でも，ある程度まとまった数のリクエストのうち，少なくとも 1 つのリクエストは滞りなく処理が行われるのが通常である．最小値に変化が生じたときは，たとえばデータベースの肥大化など，リクエストの処理内容に影響を及ぼすような原因により発生した性能異常兆候の可能性がある．

3 つめに，各々のウェブアプリケーションの性質に応じてリクエストの処理時間を管理できる．ウェブアプリケーションの修復を行う際にはリクエストの処理時間を異常発生前と同じ値に回復できることが望ましいが，原因によってはそれが難しい場合がある．たとえば，クライアント数の増加にともなってリクエストの処理時間が増大した場合，サーバの最大同時接続数を増加させることでリクエストの処理時間の最大値は回復できたとしても，中央値は逆に増大してしまうことがある．そのような場合，たとえば，すべてのユーザへサービスの最低品質を保証したいウェブアプリケーションの場合は最大値を最優先して管理しなければならないので，中央値を犠牲にして最大同時接続数を増加させる必要があると考えられる．しかし，個々のリクエストの処理時間を管理していたのでは，この調整を行うのは難しい．また，本論文では管理する統計値を上にあげた 4 種類に定めているが，ウェブアプリケーションによっては特性値としてリクエストの処理時間の 95%値や 90%値を選択できる可能性もあると考えている．

3.2　リクエストの種類ごとの管理図の作成

本論文ではリクエストの種類ごとに管理図を作成し，特性値を管理する．リクエストの種類とは処理内容によって分類するものであり，たとえば，リクエストの URL を用いて分類することができる．RUBiS では 27 種類ある．リクエストの URL を見ると，リクエストの処理に利用する Servlet の名前や，サーバに受け渡すパラメータの名前が分かる．Servlet の名前を見ることで，おおよそリクエストの種類を分類できる．たとえば，*SearchItemsByCategory* という名前の Servlet を利用していればユーザによって求められたカテゴリに存在する商品の一覧を表示するリクエストだと分かるし，*RegisterUser* という名前の Servlet を利用していれば与えられたユーザを利用者として新たに登録するリクエストだと分かる．さらに，SearchItemsByCategory という名前の Servlet を利用していたとしても，パラメータとしてカテゴリだけを受け渡すのか，それともカテゴリだけではなく地域も受け渡すのかによって，リクエストの種類を分類する．この分類はウェブアプリケーションの管理者または開発者が行う．リクエストの種類ごとに管理図を作成する目的は 2 つあり，以下で述べる．

1 つめの目的はリクエストの種類ごとに管理図を作成することで，異常の兆候をより早期に検出する可能性を高くすることである．リクエストは種類ごとに異なるコンポーネントを利用するため，処理時間の大きさもリクエストの種類ごとに異なってくる．たとえば，データベースサーバへアクセスし動的なウェブページを要求するリクエストの種類に対し，ウェブサーバのみで処理可能な静的なウェブページを要求するリクエストの種類では通常処理時間が小さくなる．このことを考慮せずにすべてのリクエストの種類の処理時間を同一の管理図で監視していたのでは，比較的処理時間の小さいリクエストの種類に発生した性能異常の兆候を見逃してしまう可能性がある．

2 つめの目的はリクエストの種類ごとに管理図を作成することで，異常兆候検出後の原因究明に有益な情報を得ることである．管理図を用いて行うのは性能異常兆候の検出までであり，その後の原因究明は人手により行う．その際，すべてのリクエストの処理時間をまとめ

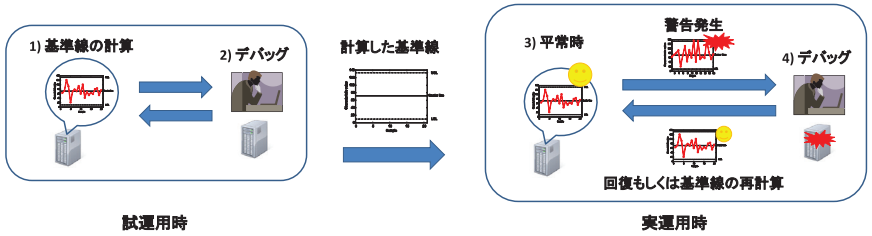


図 4　管理図の典型的な適用シナリオ
Fig. 4　A typical scenario for applying control charts.

て 1 つの管理図で監視していたのでは，管理図から得た情報から原因を探ることは難しい．しかし，リクエストは種類ごとに異なるコンポーネントを利用するため，性能異常の兆候が発生したリクエストの種類に注目することで，原因と考えられるコンポーネントを絞り込むことができる．性能異常の兆候が発生したリクエストの種類でのみ共通して利用しているコンポーネントに原因がある可能性が高いといえる．たとえば，データベース内のあるテーブルにアクセスするリクエストの種類の管理図のみ警告を発し，その他の管理図は警告を発しなかった場合，そのテーブルに性能異常の原因があるのではないかと推測することができる．

3.3　管理図の適用シナリオ

管理図は試運用時と実運用時ともにウェブアプリケーションに適用することができる．管理図の典型的な適用シナリオを**図 4** に示す．まず，試運用時に暫定的な基準線を計算する．もしこの際に管理図が警告を発したならば，原因究明を行い性能異常からの回復と基準線の再計算を行わなければならない．逆に管理図が警告を発しなければ，その時点での管理図の基準線を利用し，実運用時にリクエストの処理時間を管理していくことになる．

実運用時には，試運用時に計算した管理図の基準線を用いてリクエストの処理時間を管理していく．そして管理図が警告を発した場合はシステムになんらかの変化が生じたと考え，原因究明と回復を行う．原因究明を行った結果，もしリクエストの処理時間の変化が避けられず，また性能異常がサービスには影響を及ぼさないものと判断した場合はその時点での処理時間を利用して管理図の基準線の再計算を行うことになる．

4.　ケーススタディ

ウェブアプリケーションの性能異常兆候検出への管理図の適用可能性を示すために，管理図を利用して性能異常の兆候を検出できたケーススタディを報告する．すべてのケーススタディを通しての，管理図が警告を発したときの処理時間の平均値の増大は最大で 141 ミリ秒であり，性能異常が致命的になる前に，兆候を検出できたといえる．それぞれのケーススタディではさらに，管理図が発した警告が正しく性能異常の兆候を検出できていることを示すために，著者らが原因究明を行った．また，そのうちのいくつかのケーススタディでは，性能異常が致命的になる前にシステムの修復を行い，性能を回復することができた．

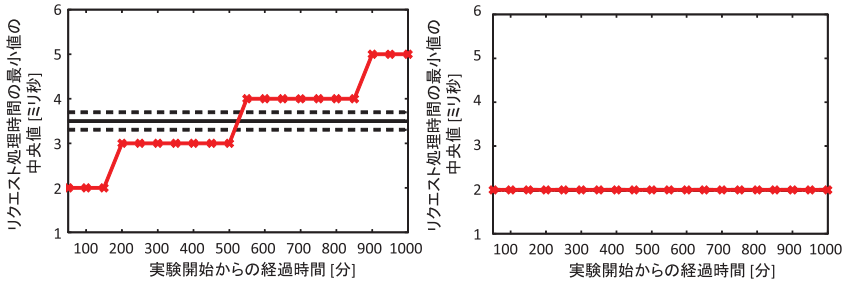
4.1　実験環境

実験では特性値としてリクエストの処理時間の平均値，最大値，中央値，最小値の 4 種類の統計値を用い，それぞれの統計値の計算は 10 分ごとに行った．ウェブアプリケーションは Java EE 上で動作する RUBiS[4] を対象とした．RUBiS はオークションサイト eBay.com[3] を模したベンチマーク用ウェブアプリケーションである．この RUBiS のリクエストの種類は 27 種類あり，その例は *Home*，*SearchItemsInCategory*，*PutBid*，*RegisterUser* である．

実験は RUBiS に付属しているクライアントエミュレータに若干の変更を加え行った．加えた変更はクライアントの動作をより人間のユーザに近付けるためのものであり，各クライアントスレッドの開始間隔を 5 秒間にし，各クライアントスレッドに 8 秒間のタイムアウトを設けた．RUBiS に付属しているクライアントエミュレータは実験開始時にいっせいにクライアントスレッドを作成し，ウェブアプリケーションへアクセスを行う．しかし，この動作により，一時的にサーバの負荷が増大し，さらにこの影響が実験を通して残ってしまうことがある．この挙動を避けるために，各クライアントスレッドの開始間隔を 5 秒間にした．また，RUBiS に付属しているクライアントエミュレータはリクエスト送信後，サーバから応答があるまで待ち続ける．しかし，人間のユーザではそのような挙動は通常望めないため，通常人間が応答を待つ時間といわれている 8 秒間のタイムアウトを各クライアントスレッドに設けた．この 8 秒間という値は他の研究の実験でも採用されている値である[6]．

3.1 節で，Apache を利用したシステムではサーバ側で各リクエストの処理時間を測定できることを述べたが，本論文の実験では RUBiS のクライアントエミュレータの機能を用いてクライアント側で各リクエストの応答時間を測定し，それをサーバ側で測定した各リクエストの処理時間として代用している．しかし本論文の実験はサーバとクライアントを同一 LAN 内に配置し行ったため，クライアント側で測定した各リクエストの応答時間をサーバ側で測定した各リクエストの処理時間として代用する影響はほとんどないと考えられる．

実験は 3.00 GHz の CPU，2 GB のメモリを搭載している 4 台のマシンを利用して行った．それぞれのマシンでは RUBiS のクライアントエミュレータ，ウェブサーバ Apache 2.2.11,



(a) インデックス追加前　(b) インデックス追加後

図 5　基準線計算時の SearchItemsInCategory-Minimum 管理図．黒い実線は中心線，黒い破線は管理限界線である．図 5 (b) は基準線がないように見えるが，実際は 4 本が重なっている
Fig. 5　SearchItemsInCategory-Minimum control charts when calculating baselines.

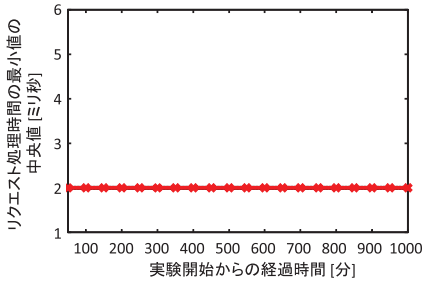


図 6　基準線計算時の SearchItemsInRegion-Minimum 管理図．基準線がないように見えるが，実際は 4 本の線が重なっている
Fig. 6　SearchItemsInRegion-Minimum control chart when calculating baselines.

アプリケーションサーバ JBoss 5.0.1，データベースサーバ MySQL 5.1.34 を動作させた．オペレーティングシステムは Linux であり，カーネルのバージョンは 2.6.27 である．

ここで，ケーススタディの説明のために管理図の名前規則を説明する．それぞれの管理図は *request-type-statics* 管理図というように名前を付ける．*request-type* はリクエストの種類を指し，*statics* は統計値を指す．たとえば，SearchItemsInCategory-Minimum 管理図とは SearchItemsInCategory というリクエストの種類の，リクエストの処理時間の最小値を管理するための管理図である．

4.2　`items` テーブルの肥大化

まずは RUBiS に対してクライアント数 200 の負荷をかけ，各リクエストの処理時間を測定し，管理図の基準線を計算した．その際，いくつかの管理図が警告を発した．本節ではそのうちの 1 つの管理図について詳しく説明する．

本節で注目するのは SearchItemsInCategory-Minimum 管理図である．この管理図は**図 5** (a) のように，すべての打点が管理限界線の外側にあり，管理図が警告を発している．さらに図 5 (a) を見ると，時間が経過するに従ってリクエスト処理時間の最小値の中央値が増大していることが分かる．このとき，SearchItemsInCategory の処理時間の最小値の中央値の増大は数ミリ秒であるが，今後致命的な性能異常に発展する可能性があり，原因を究明し対策を行う必要がある．

ここからは，管理図が発した警告が実際にウェブアプリケーションの性能異常の兆候だったことを示すために，著者らが行った原因究明を述べる．まず管理図による警告を受けて，著者らが管理図から得られた情報を利用して考えられる原因の絞り込みを行った．すべての管理図を調査した結果，最小値の中央値が増大した結果警告を発しているのは SearchItemsInCategory の管理図のみであった．よって，図 5 (a) の管理図が発した警告の原因は SearchItemsInCategory に特有なものではないかと考えられる．

さらに，著者らは RUBiS のソースコードを用いて SearchItemsInCategory が利用しているコンポーネントを調べ，発せられた警告の原因となっている可能性のあるコンポーネントを絞り込んだ．SearchItemsInCategory と同様の処理を行う SearchItemsInRegion に注目することで，原因と考えられるコンポーネントの絞り込みを速やかに行うことができた．SearchItemsInRegion-Minimum 管理図は**図 6** のように警告を発していない．SearchItemsInCategory が利用し，かつ SearchItemsInRegion が利用していないコンポーネントは SearchItemsByCategory サーブレット，SB_SearchItemsByCategory セッションビーン，Query セッションビーンによって発行される SQL クエリの 3 つであり，この 3 つのコンポーネントに原因があると考えられる．

さらに著者らは詳細に原因究明を行った．著者らは 3 つのコンポーネントのうち，複雑な SQL クエリをデータベースへ発行する Query セッションビーンを最初に疑い調査した．Query セッションビーンよって発行される SQL クエリは `SELECT items.id FROM items WHERE items.category=?  AND items.end_date>=NOW() ORDER BY items.end_date ASC LIMIT ?, ?;` であり，?はユーザによって与えられる変数である．この SQL クエリを見ると，アクセスするのは `items` テーブルだけであり，`items` テーブルを調べる必要がある．`items` テーブルを調べた結果，RUBiS では `items` テーブルからレコードが削除されること

表 1　クライアント数が 300 のときの管理図の結果．KeepAliveTimeout 変更前と変更後の管理図の結果．108 は 1 つのリクエストの種類の管理図での打点数 4 とすべてのリクエストの種類の数 27 をかけた値である
Table 1　Result of control charts when the number of clients is 300.

		警告数 /打点数	警告が発せられたリクエストの種類の数 /すべてのリクエストの種類の数
変更前	*-Average 管理図	91/108	26/27
	*-Maximum 管理図	91/108	25/27
変更後	*-Average 管理図	0/108	0/27
	*-Maximum 管理図	2/108	2/27

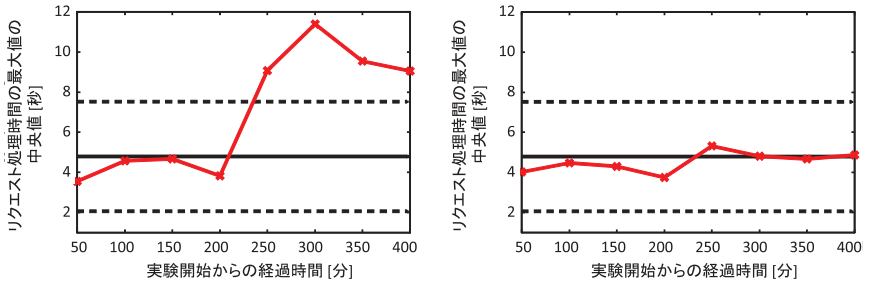
はなく，商品が追加されるにつれて items テーブルは肥大化し続けることが分かった．この items テーブルの肥大化により，SearchItemsInCategory の処理時間の最小値の中央値が増大したと考えられる．

さらに原因究明を続けた結果，SearchItemsInCategory の処理時間の最小値の中央値の増大から回復するためには，items テーブルに category と end_date からなるインデックスを追加すれば良いということが分かった．このインデックスを追加することで，各クエリに対して該当レコードの絞り込みが迅速に行えるようになり，テーブルの肥大化による処理時間の増大が軽減できる．この変更を行い再実験した結果が図 5 (b) の SearchItemsInCategory-Minimum 管理図である．管理図は警告を発しなくなり，items テーブルが肥大化した後も SearchItemsInCategory の処理時間の最小値の中央値は一定を保っている．よって，管理図は性能異常の兆候を検出できていたといえる．また，インデックスの追加による副作用は見られず，性能異常から回復できた．

4.3　クライアント数の 200 から 300 への増加

基準線を計算した後，クライアント数を 200 から 300 へ増加させる実験を行った．その結果，クライアント数が 300 に増加した後に*-Average 管理図と*-Maximum 管理図が多数の警告を発した．ここで，*はワイルドカードである．その結果を示したのが**表 1** である．表 1 に示した警告はすべて上方管理限界線を超えた打点によるものだった．このときの典型的な管理図を**図 7** (a) に示す．このとき，クライアント数が 200 から 300 に増加したことによる全リクエストの処理時間の平均値の増大は 141 ミリ秒であったが，今後致命的な性能異常に発展する可能性があり，原因を究明し対策を行う必要がある．

ここからは，管理図が発した警告が実際にウェブアプリケーションの性能異常の兆候だったことを示すために，著者らが行った原因究明を述べる．表 1 を見ると，ほぼすべてのリクエストの種類で管理図が警告を発していることが分かる．このことから，管理図が発した



(a) KeepAliveTimeout 変更前　(b) KeepAliveTimeout 変更後

図 7　クライアント数を 200 から 300 に増加させたときの AboutMe-Maximum 管理図．黒い実線は中心線，黒い破線は管理限界線である．実験開始からの経過時間が 200 分のときに 100 のクライアントスレッドの起動を開始した．図 7 (a) が KeepAliveTimeout を 1 に変更する前の管理図であり，図 7 (b) が 1 に変更した後の管理図である
Fig. 7　AboutMe-Maximum control charts when increasing the number of clients from 200 to 300.

警告はすべてのリクエストの種類に共通する要因によるものであると推測できる．よって，著者らは管理図から得た情報を利用して，原因はすべてのリクエストが通過するフロントエンドのウェブサーバにあると推測した．

さらに著者らは Apache ウェブサーバの設定に注目し，詳細に原因究明を行った．まず初めに注目したのが MaxClients である．MaxClients はウェブサーバが同時に扱えるリクエストの上限数を決定する設定である．一般に，MaxClients はウェブサーバの性能に大きな影響を及ぼすことが知られている．しかし，MaxClients を増加させても管理図は警告を発し，MaxClients の設定が管理図が発した警告の原因でないことが分かった．次に，Apache ウェブサーバがリクエスト終了後維持するコネクションのタイムアウトの設定である KeepAliveTimeout に注目した．KeepAliveTimeout も MaxClients 同様，ウェブサーバの性能に大きな影響を及ぼすことが一般に知られている．KeepAliveTimeout をデフォルトの 5 秒から 1 秒に設定し直したところ，表 1 に示したように，ほぼすべての管理図は警告を発しなくなり，クライアント数が 300 に増加した後もリクエストの処理時間の平均値の中央値と最大値の中央値は管理限界線の内側に打点された．そのときの管理図の 1 つを図 7 (b) に示した．よって，管理図は性能異常の兆候を検出できていたといえる．また，KeepAliveTimeout の変更による副作用は見られず，性能異常から回復できた．

4.4　クライアント数の 300 から 400 への増加

次に，クライアント数を 300 から 400 へ増加させる実験を行った．この実験では，クラ

表 2　クライアント数が 400 のときの管理図の結果．20，16，72 はそれぞれ，各管理図の打点数 4 とリクエストの種類の数 5，4，18 をかけた値である
Table 2　Result of control charts when the number of clients is 400.

	*-Maximum 管理図		
	ウェブ	アプリケーション	データベース
警告数 /打点数	0/20	4/16	43/72
警告が発せられたリクエストの種類の数 /すべてのリクエストの種類の数	0/5	2/4	17/18

イアント数が 400 に増加した後に*-Maximum 管理図が多数の警告を発し，それらはすべて上方管理限界線を超える打点によるものだった．このとき，クライアント数の 300 から 400 への増加にともなう全リクエストの処理時間の平均値の増大は 91 ミリ秒であったが，今後致命的な性能異常に発展する可能性があり，原因を究明し対策を行う必要がある．

ここからは，管理図が発した警告が実際にウェブアプリケーションの性能異常の兆候だったことを示すために著者らが行った原因究明を述べる．まず RUBiS のソースコードを調査し，管理図の結果をまとめたのが**表 2** である．実験環境は 3 層構造で構成しており，リクエストの種類は処理が最終的に及ぶサーバによって 3 種類に分類できる．調査の結果，それぞれに分類されるリクエストの種類の数は 5，4，18 であることが分かった．表 2 に示したように，処理がウェブサーバにしか及ばないリクエストの種類の管理図は警告を発しておらず，警告を発しているのは処理がアプリケーションサーバに及ぶリクエストの種類であった．よって，著者らは管理図から得た情報を利用して，RUBiS の性能異常の原因はアプリケーションサーバの JBoss にあると推測した．

JBoss に焦点を当てて著者らはさらに詳細に原因究明を行った．JBoss のログファイルを調査した結果，JBoss が Apache からリクエストを受け取るために生成されたスレッド数が上限に達していることが分かった．上限を設定する `maxThreads` をデフォルトの 200 から 400 に増加させ実験したところ，すべての管理図は警告を発しなくなった．このことから，管理図はウェブアプリケーションの性能異常の兆候を検出できていたといえる．また，`maxThreads` を 200 から 400 へ増加させたことによる他の管理図への副作用はなく，性能の回復を行うことができたといえる．

4.5　クライアント数の 400 から 500 への増加

最後に，クライアント数を 400 から 500 へ増加させる実験を行った．この実験では，クライアント数が 500 に増加した後に*-Average 管理図と*-Maximum 管理図が多数警告を発した．結果をまとめたのが**表 3** である．クライアント数を 400 から 500 へ増加させたときの全リクエストの処理時間の平均値の増大は 105 ミリ秒であったが，今後致命的な性能異常に発展する可能性があり，原因を究明し対策を行う必要がある．

表 3　クライアント数を 400 から 500 へ増加させたときの管理図の結果．`KeepAlive` 変更前と変更後の管理図が発した警告数と管理図が警告を発したリクエストの種類の数を統計値ごとに表にしている．さらに，上方管理限界線と下方管理限界線のどちらを超えた打点による警告だったかも分類している．216 はクライアント数が 400 と 500 のときを合わせた管理図の打点数 8 とすべてのリクエストの種類の数の 27 をかけた値である．これまでと違い，この実験ではクライアント数が 400 のときの結果も重要なため，示した
Table 3　Result of control charts when increasing the number of clients from 400 to 500.

			警告数 /打点数	警告が発せられたリクエストの種類の数 /すべてのリクエストの種類の数
変更前	*-Average	上方管理限界線	44/216	20/27
		下方管理限界線	0/216	0/27
	*-Maximum	上方管理限界線	49/216	21/27
		下方管理限界線	0/216	0/27
	*-Median	上方管理限界線	37/216	7/27
		下方管理限界線	0/216	0/27
変更後	*-Average	上方管理限界線	0/216	0/27
		下方管理限界線	160/216	20/27
	*-Maximum	上方管理限界線	0/216	0/27
		下方管理限界線	206/216	26/27
	*-Median	上方管理限界線	84/216	12/27
		下方管理限界線	0/216	0/27

ここからは，管理図が発した警告が実際に RUBiS の性能異常の兆候であったことを示すために著者らが行った原因究明を述べる．この実験では処理がウェブサーバにしか及ばないリクエストの種類から処理がデータベースサーバにまで及ぶリクエストの種類まで 3 種類すべてで管理図が警告を発した．よって，著者らは管理図が発した警告から得た情報から，クライアント数を 300 に増加させた 4.3 節での実験同様，フロントエンドであるウェブサーバに警告の原因があると推測した．

著者らはさらに詳細に原因究明を行った．今回も 4.3 節での実験同様，まず初めに `MaxClients` をデフォルトの 256 から 512 へ増加させて実験を行った．この変更により，クライアント数が 500 のときの管理図が発した警告は減少した．しかし，副作用として，クライアント数が 400 のときに管理図は警告を発してしまい，`MaxClients` は管理図が発した警告の原因ではなかったといえる．

次に，4.3 節同様，もう一度 `KeepAliveTimeout` に注目した．すでに 4.3 節で

KeepAliveTimeout を 1 秒に設定したので，今回は KeepAlive をオフにして実験を行った．その結果を表 3 に示した．今回の結果では*-Average，*-Maximum，*-Median 管理図が多数の警告を発した．しかし，その結果を注意深く見ると，KeepAlive がオンのときの結果とは異なる結果であった．KeepAlive がオンのときに管理図が発した警告はすべて上方管理限界線を超える打点によるものだったのに対し，KeepAlive がオフのときに*-Average 管理図と*-Maximum 管理図が発した警告は下方管理限界線を超える打点によるものだった．

実験結果を詳しく調査した結果，*-Average 管理図と*-Maximum 管理図が発した警告は，性能の向上により処理時間が小さくなった結果発せられた警告であることが分かった．しかし，その副作用として，*-Median 管理図はクライアント数が 400 のときにも上方管理限界線を超える打点による警告を発してしまった．ところが，調査の結果，KeepAlive をオフにしたことによるリクエストの処理時間の中央値の増大は数ミリ秒であり，平均値を約 90 ミリ秒減少させることを優先して，KeepAlive をオフにすることがよいと考えられる．このことから，管理図はウェブアプリケーションの性能異常の兆候を検出できていたといえる．今回の実験のようにリクエストの処理時間が大幅に変化した場合には，それまでの管理図でその先もリクエストの処理時間を管理していくことはできないため，基準線の再計算を行う必要がある．

4.6　RegisterUser における 2 つの実行パス

管理図は処理時間の統計的な変化を検出し警告を発するため，ウェブアプリケーションの作成者が意図した正常な動作でも処理時間が変化すると警告を発してしまう．本節では，基準線を計算する際に管理図が発したそのような警告の例をあげ，説明する．**図 8** は警告を発した RegisterUser-Minimum 管理図である．この管理図はリクエスト処理時間の最小値の中央値が減少していく傾向を見せた唯一の管理図であるため，この警告は RegisterUser に固有のものであると推測できる．

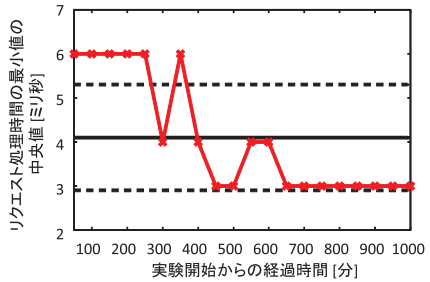


図 8　正常な動作に対して警告を発してしまった RegisterUser-Minimum 管理図．黒い実線は中心線であり，黒い破線は管理限界線である
Fig. 8　RegisterUser-Minimum control chart which raised warnings against for normal behavior

著者らが RUBiS のソースコードを調査した結果，RegisterUser には正常時に通過する実行時のパスが 2 つあることが分かった．ここで，RegisterUser は与えられたユーザを新たにメンバに登録するリクエストの種類である．1 つの実行パスはユーザを正常に登録できたときに通過するパスである．もう 1 つの実行パスは入力されたユーザ名がすでに使用されており，ユーザを登録できなかったときに通過するパスである．前者の実行パスはユーザをデータベースへ登録するための処理を行うのに対し，後者の実行パスはユーザをデータベースへ登録せずに応答するため，処理時間は後者の実行パスを通過したときの方が小さくなる傾向にある．しかし，後者は想定されている例外であり，性能異常ではないといえる．

この挙動により RegisterUser の処理時間の最小値の中央値が時間の経過に従って減少していくのは，RUBiS のクライアントエミュレータが原因である．RUBiS のクライアントエミュレータは新規にユーザを登録するときに，ユーザ名として乱数を用いて生成した整数を使用する．その結果，実験開始から時間が経過するに従ってすでに登録されているユーザ名を利用する確率が高くなり，後者の実行パスを通過する可能性が高くなる．また，この挙動は RUBiS を実運用した際にも起こる可能性がある挙動だといえる．本節では管理図が正常時の動作に対し警告を発してしまった例をあげたが，無意味に警告を発しているわけではなく，管理図のウェブアプリケーションの性能異常の兆候検出への適用可能性を否定する結果ではないと考えている．

5.　管理図が発した警告に対する考察

本論文では RUBiS に発生した性能異常兆候の検出後，人手で原因究明を行った．本論文では管理図をリクエストの種類ごとかつリクエストの処理時間の統計値ごとに作成しているため，性能異常の兆候を検出すると同時に性能異常の兆候が発生しているリクエストの種類とリクエストの処理時間の統計値が分かる．著者らは原因究明の際にこの情報を利用した．

ただし，複数のリクエストの種類の管理図が同時期に警告を発した場合は，原因究明の際に注意すべき点が 2 つある．1 つめに，まず初めに注目すべきなのは，長期間にわたって管理図が警告を発しているリクエストの種類だということである．これは，一時的な性能異常に惑わされることなく，より重要な性能異常に注目するために注意すべき点である．その一例が図 5 (a) に示した SearchItemsInCategory-Minimum 管理図である．この管理図はす

べての打点が管理限界線を超えている．このように恒常的な性能異常にいち早く注目できた．2 つめに，長期間にわたって管理図が警告を発しているリクエストの種類の中でも，特にいち早く管理図が警告を発しているリクエストの種類に注目すべきだということである．性能異常は伝播していくことがあり，その中で一番最初に性能異常の兆候が発生したリクエストの種類を特定することで，性能異常の兆候の発生源を特定することができる．特に，4.3 節〜4.5 節でのケーススタディのように，コネクション管理が原因の性能異常ではこの注意事項が重要になる．たとえば，アプリケーションサーバとデータベースサーバ間のコネクションが不足した場合，時間が経過するにつれて処理待ちのリクエストがアプリケーションサーバに滞積していく．その結果として，ウェブサーバとアプリケーションサーバ間のコネクションも不足するといった事態が想定できる．

また，ソースコードを利用してシステムの構造を知ることで，さらに考えられる原因を絞り込むことができる．先に述べたように，性能異常が発生しているリクエストの種類が利用しているコンポーネントを特定することで，原因と考えられるコンポーネントを絞り込むことができる．しかし，その際に気をつけなければならないことがある．それは，たとえあるリクエストの種類が，性能異常の原因となっているコンポーネントを利用していたとしても，そのリクエストの種類に必ず性能異常が表れるとは限らないということである．たとえば，あるコンポーネントに，同時に処理できる上限数を超える数のリクエストが到達した場合，一部のリクエストのみそのコンポーネントのための待ち行列に入れられる．この場合，性能異常が発生するのはその，待ち行列に入れられた一部のリクエストのみであり，他のリクエストも同じコンポーネントを利用しているが，そのリクエストには性能異常は発生しないことがある．これは 4.5 節のケーススタディに顕著に表れている．表 3 を見ると，フロントエンドであるウェブサーバに性能異常兆候の原因があったにもかかわらず，すべてのリクエストの種類の管理図が警告を発しているわけではない．

6. 関 連 研 究

ウェブアプリケーションで発生した性能異常の検出と原因特定を行うために有用な情報を収集し，ウェブアプリケーションの管理者に提示する研究が行われている．有用な情報とは，各リクエストが通過した実行時のパスや，リクエストを処理する際に各コンポーネントに滞在した時間やそのときの資源利用率などである．これらの情報を見ることで，管理者は他のコンポーネントと比較して明らかに滞在時間が長かったり，CPU 利用率が高かったりするコンポーネントを発見することができる．その後管理者はその疑わしいコンポーネントに注目して性能異常の原因究明を行うことができる．

これらの研究はどのような情報を収集すべきか，またどのように情報を収集すべきかといったことに主眼をおいている．したがって，これらの研究では収集した情報をウェブアプリケーションの管理者へ提示することまで行い，その後性能異常の検出は管理者にまかせる．Aguilera らの研究[7] と Whodunit[8] は収集する情報はお互いに異なるものの，どちらも収集した値の平均値を，リクエスト実行時のパスをグラフ化したものに付随して提示することまでを行い，性能異常の検出は管理者が判断する．また，Chen らの研究[9] では収集した情報を箱ひげ図などを用いて，値の平均値だけではなく分散にも注目できるようにし，管理者がより理解しやすい形式で提示している．管理図を，これらの研究により得た情報に適用することで，人間が判断するよりも早期に性能異常の兆候を検出できる可能性がある．また，管理図が適用できれば，各管理者の能力に依存せずに性能異常の兆候を検出できる．

また，本論文とは異なったアプローチで性能異常の検出と原因究明を行う研究も存在する．これらの研究はアクセス数や資源利用率，システムのログに生じる変化を性能異常として検出する．著者らは，性能を表す指標として用いられることの多い処理時間に生じる変化から性能異常を検出する手法も必要だと考えている．また，これらの関連研究の提案手法で検出できる性能異常と本論文で適用した手法で検出できる性能異常は異なると考えており，補完し合う関係にあるとも考えている．

Bodik らの研究[10] はユーザが性能異常に遭遇した場合にはユーザの動作が通常時と変わることが多いという知見に基づいて，性能異常を検出している．監視対象のウェブアプリケーションの，一定時間ごとの各ウェブページへのアクセス数をベクトルとして記録し，そのベクトルの変化を検出する．ベクトルに生じた変化の検出には χ^2 検定と単純ベイズ分類を利用する．Magpie[11] は一部のユーザにのみ発生するような局所的な異常を検出する．各リクエストを処理する際に発行されたイベントと，資源利用率を収集する．そして，収集した情報をクラスタリングし，クラスタからはずれたリクエストを見つけることでウェブアプリケーションの性能異常を検出する．Cohen らの研究[12] は資源の利用率に変化が表れるような性能異常を対象としている．マシンの資源利用率と管理者が定めた性能指標の値を一定時間ごとに測定し，測定したデータに Tree-Augmented Naive Bayesian networks を適用して性能異常の原因となっている資源の挙動を発見する．たとえば，リクエストの処理時間の平均値が 1 秒を超えた原因は CPU 利用率が 50%を超えたことである，といった具合である．また，このように性能異常が発生する可能性が高い資源の挙動を事前に把握しておくことで，性能異常の兆候を検出できる．さらに，システムのログからウェブアプリケーショ

ンの性能異常を検出する研究も存在する[13)–15)]．これらの研究は出力されたシステムのログに生じた変化をデータマイニングと統計的手法を用いて検出し，それを性能異常の兆候だとして管理者に提示する．

また，ウェブアプリケーションに管理図を適用した研究としては Ye らの研究[2)] がある．ただし，この研究では管理図をウェブアプリケーションの性能異常の検出ではなく，ウェブアプリケーションへの侵入検知に適用している．この研究ではサーバで発行されるイベント数を特性値として管理図を作成し，イベント数に生じる変化により発せられた警告を侵入検知とする．

現在，ウェブアプリケーションの性能異常回避のためにさまざまな研究が行われている[16)–24)]．これらの手法は，パラメータ設定など，それぞれの研究が個々の性能異常の原因に対処するものである．本論文のように管理図で性能異常を検出し，原因究明が行えたあとにその箇所に適用することができると考えている．

7. ま と め

ウェブアプリケーションの性能異常が重要な問題となりつつあり，性能異常の発生をいち早く検出することが求められている．性能異常の兆候を検出できれば，性能異常がユーザに気づかれるほど致命的になる前に対策を行える可能性が増す．

本論文では管理図という統計的手法を利用して，ウェブアプリケーションの性能異常兆候の検出を行った．管理図は過去の品質と現在の品質のずれを統計的に判定するものであるため，管理図を用いてウェブアプリケーションの過去の処理時間と現在の処理時間を比較することで，処理時間に表れる性能異常の兆候を検出できる．

管理図をウェブアプリケーションの性能異常兆候の検出に適用するために，2 つの工夫を行った．1 つめの工夫は管理対象の値として個々のリクエストの処理時間ではなく，一定時間ごとにリクエストの処理時間の統計値を 4 種類（平均値，最大値，中央値，最小値）計算し，それらを選択することである．この工夫により，管理図をリクエストの処理時間の正常時の揺らぎに対して堅牢にし，リクエストの処理時間にさまざまな形で表れる性能異常を検出可能にし，また管理者が個々のウェブアプリケーションの性質に応じてリクエストの処理時間を管理することを可能にした．2 つめの工夫は管理図をウェブアプリケーション全体で 1 つではなく，リクエストの種類ごとに作成することであり，性能異常兆候のより早期の検出と性能異常兆候検出後の原因究明に有益な情報を得ることを可能にした．

管理図を用いて，オークションサイトを模したベンチマーク用ウェブアプリケーションである RUBiS に発生した性能異常の兆候を 4 つ検出した．実験では，管理図が警告を発した後人手で原因究明を行い，確かに性能異常の兆候であることを確認した．検出した性能異常の兆候は，(1) サービス運用の経過にともなう，データベース内のテーブルへのインデックス不足によるリクエストの処理時間の増大と，(2) クライアント数の増加にともない性能パラメータの設定が不適切になってしまったことによるリクエストの処理時間の増大に大別でき，(1) はデータベース内のテーブルへインデックスを追加することで，(2) は Apache の `keepAlive` に関するパラメータと JBoss の `maxThreads` を適切に設定し直すことでそれぞれ性能異常が致命的になってしまうのを未然に防ぐことができた．

参 考 文 献

1) 中村達男：管理図の作り方と活用，日本規格協会 (1999).
2) Ye, N., Borror, C. and Zhang, Y.: EWMA Techniques for Computer Intrusion Detection Through Anomalous Changes in Event Intensity, *Quality and Reliability Engineering International*, Vol.18, No.6, pp.443–451 (2002).
3) eBay.com. http://www.ebay.com/
4) RUBiS: Rice University Bidding System. http://rubis.objectweb.org/
5) 鐵　健司：新版　品質管理のための統計的方法入門，日科技連出版社 (2000).
6) Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G. and Fox, A.: Microreboot — A Technique for Cheap Recovery, *Proc. 6th Symposium on Operating Systems Design and Implementation* (*OSDI 2004*), San Francisco, California, USA, USENIX, pp.31–44 (2004).
7) Aguilera, M.K., Mogul, J.C., Wiener, J.L., Reynolds, P. and Muthitacharoen, A.: Performance Debugging for Distributed Systems of Black Boxes, *Proc. 19th Symposium on Operating Systems Principles* (*SOSP 2003*), Bolton Landing, New York, USA, pp.74–89, ACM (2003).
8) Chanda, A., Cox, A.L. and Zwaenepoel, W.: Whodunit: Transactional Profiling for Multi-Tier Applications, *Proc. 2nd European Conference on Computer Systems* (*EuroSys 2007*), Lisbon, Portugal, ACM SIGOPS/EuroSys, pp.17–30 (2007).
9) Chen, M.Y., Accardi, A., Kiciman, E., Lloyd, J., Patterson, D., Fox, A. and Brewer, E.: Path-Based Failure and Evolution Management, *Proc. 1st Symposium on Networked Systems Design and Implementation* (*NSDI 2004*), San Francisco, California, USA, USENIX, pp.309–322 (2004).
10) Bodik, P., Friedman, G., Biewald, L., Levine, H., Candea, G., Patel, K., Tolle, G., Hui, J., Fox, A., Jordan, M.I. and Patterson, D.: Combining Visualization and Statistical Analysis to Improve Operator Confidence and Efficiency for Failure Detection and Localization, *Proc. 2nd International Conference on Autonomic Com-*

puting (*ICAC 2005*), Seattle, Washington, USA, pp.89–100, IEEE (2005).
11) Barham, P., Donnelly, A., Isaacs, R. and Mortier, R.: Using Magpie for request extraction and workload modellingn, *Proc. 6th Symposium on Operating Systems Design and Implementation* (*OSDI 2004*), San Francisco, California, USA, USENIX, pp.259–272 (2004).
12) Cohen, I., Goldszmidt, M., Kelly, T., Symons, J. and Chase, J.S.: Correlating instrumentation data to system states: A building block for automated diagnosis and control, *Proc. 6th Symposium on Operating Systems Design and Implementation* (*OSDI 2004*), San Francisco, California, USA, USENIX, pp.231–244 (2004).
13) Yamanishi, K. and Maruyama, Y.: Dynamic Syslog Mining for Network Failure Monitoring, *Proc. 11th International Conference on Knowledge Discovery in Data Mining* (*KDD 2005*), Chicago, Illinois, USA, ACM SIGKDD, pp.499–508 (2005).
14) Lim, C., Singh, N. and Yajnik, S.: A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems, *Proc. Annual International Conference on Dependable Systems and Networks* (*DSN 2008*), Anchorage, Alaska, USA, IEEE/IFIP, pp.398–403 (2008).
15) Xu, W., Huang, L., Fox, A., Patterson, D. and Jordan, M.I.: Detecting Large-Scale System Problems by Mining Console Logs, *Proc. 22nd symposium on Operating systems principles* (*SOSP 2009*), Big Sky, Montana, USA, pp.117–132, ACM (2009).
16) Xi, B., Liu, Z., Raghavachari, M., Xia, C.H. and Zhang, L.: A Smart Hill-Climbing Algorithm for Application Server Configuration, *Proc. 13th International World Wide Web Conference* (*WWW 2004*), New York, New York, USA, pp.287–296 (2004).
17) Sugiki, A., Kono, K. and Iwasaki, H.: Tuning mechanisms for two major parameters of Apache web servers, *Software: Practice and Experience*, Vol.38, pp.1215–1240 (2008).
18) Zheng, W., Bianchini, R. and Nguyen, T.D.: Automatic Configuration of Internet Services, *Proc. 2nd European Conference on Computer Systems* (*EuroSys 2007*), Lisbon, Portugal, ACM SIGOPS/EuroSys, pp.219–229 (2007).
19) Osogami, T. and Kato, S.: Optimizing System Configurations Quickly by Guessing at the Performance, *Proc. 2007 International Conference on Measurement and Modeling of Computer Systems* (*SIGMETRICS 2007*), San Diego, California, USA, ACM SIGMETRICS, pp.145–156 (2007).
20) Stewart, C., Zhong, M., Shen, K. and O'Neill, T.: Comprehensive Depiction of Configuration-dependent Performance Anomalies in Distributed Server Systems, *Proc. 2nd Workshop on Hot Topics in System Dependability* (*HotDep 2006*), Seattle, Washington, USA, USENIX, pp.1–1 (2006).
21) Nagaraja, K., Oliveira, F., Bianchini, R., Martin, R.P. and Nguyen, T.D.: Understanding and Dealing with Operator Mistakes in Internet Services, *Proc 6th Symposium on Operating Systems Design and Implementation* (*OSDI 2004*), San Francisco, California, USA, USENIX, pp.61–76 (2004).
22) Stewart, C., Kelly, T., Zhang, A. and Shen, K.: A Dollar from 15 Cents: Cross-Platform Management for Internet Services, *Proc. 2008 Annual Technical Conference* (*USENIX 2008*), Boston, Massachusetts, USA, USENIX, pp.199–212 (2008).
23) Stewart, C. and Shen, K.: Performance Modeling and System Management for Multi-component Online Services, *Proc. 2nd Symposium on Networked Systems Design and Implementation* (*NSDI 2005*), Boston, Massachusetts, USA, USENIX, pp.71–84 (2005).
24) Stewart, C., Kelly, T. and Zhang, A.: Exploiting Nonstationarity for Performance Prediction, *Proc. 2nd European Conference on Computer Systems* (*EuroSys 2007*), Lisbon, Portugal, ACM SIGOPS/EuroSys, pp.31–44 (2007).

付録　管理図の数理

一般的な管理図では，特性値の統計値の期待値を中心線とし，その上下に特性値の統計値の標準偏差の 3 倍の幅で管理限界線をひくことに定められている（JIS Z 9021;1998）．よって，管理限界線は

$$UCL,\ LCL = E[X] \pm 3D[X]$$

となる．ここで，X は特性値の統計値（例：平均値，中央値，不良率），$E[X]$ は X の期待値，$D[X]$ は X の標準偏差である．

このとき，特性値の統計値 X が正規分布に従い，工程が管理状態にあるならば，特性値の統計値 X は管理限界線内に 99.7%，管理限界線外に 0.3%が分布する．よって，特性値の統計値が管理限界線外に打点されれば，ほとんど間違いなく工程に異常が発生していると結論づけることができる．さらに，2 章で述べたように管理限界線の内側に特性値の統計値が打点されていたとしても，その打点のされ方に特定のクセが見受けられた場合は異常が発生していると結論づけ管理図は警告を発する．本論文ではこのクセを判定するルールの数理については詳細に説明せず，説明は参考文献 1)，5) などにまかせる．

管理図は特性値の統計値 X に応じてさまざまな種類が存在するが，ここでは最も一般的な管理図である $\overline{X}$ 管理図の数理を説明する．その後で，メディアン管理図の数理についても補足する．その他の管理図の数理については参考文献 1)，5) などを参照してほしい．

$\overline{X}$ 管理図は特性値 X の平均値 $\overline{X}$ に注目して管理限界線を定める管理図であるため，管理限界線は

$$UCL,\ LCL = E[\overline{X}] \pm 3D[\overline{X}]$$

となる．

ここで，平均 μ，標準偏差 σ の特性値 X の分布からとった，大きさ n の群の平均値 $\overline{X}$ の分布は平均 μ，標準偏差 $\sigma/\sqrt{n}$ であり，中心極限定理により $\overline{X}$ の分布の形は n が大きくなるほど正規分布に近くなる．実用的には n が 4 くらい以上なら正規分布であるとしても差し支えない[1)]．ここで，特性値 X の分布の形によらず，特性値の平均値 $\overline{X}$ の分布の形が正規分布に近くなることに注意してほしい．これらの値を代入すると，管理限界線は

$$UCL,\ LCL = \mu \pm 3\frac{\sigma}{\sqrt{n}}$$

となる．

さらに，一般に μ と σ は未知であるため，大きさ n の群の平均値 $\overline{X}$ と範囲 R を用いて推定すると

$$\mu \leftarrow \overline{\overline{X}}$$

$$\sigma \leftarrow \frac{\overline{R}}{d}$$

となる．ただし，$\overline{\overline{X}}$ は（2 章で述べたように，管理図では通常 20 個の）$\overline{X}$ の平均値，$\overline{R}$ は（2 章で述べたように，管理図では通常 20 個の）R の平均値，d は群の大きさ n によって一意に定まる値である．

よって，$\overline{X}$ 管理図の管理限界線は

$$UCL,\ LCL = \overline{\overline{X}} \pm 3\frac{\overline{R}}{d\sqrt{n}} = \overline{\overline{X}} \pm A_1\overline{R}$$

となる．ただし，群の大きさ n によって一意に定まる値を $3/d\sqrt{n} = A_1$ とした．

本論文では $\overline{X}$ 管理図ではなくメディアン管理図を用いる．$\overline{X}$ 管理図が特性値 X の平均値 $\overline{X}$ に注目して管理限界線を定める管理図なのに対し，メディアン管理図は特性値 X の中央値 Me に注目して管理限界線を定める管理図である．特性値 X の分布が平均 μ，標準偏差 σ だとすると，大きさ n の群の中央値 Me の期待値と標準偏差はそれぞれ

$$E[Me] = \mu$$

$$D[Me] = \frac{m\sigma}{\sqrt{n}}$$

となる．ただし，m は群の大きさ n によって一意に定まる値である．さらに，μ と σ は大きさ n の群の中央値 Me と範囲 R を用いて推定し，

$$\mu \leftarrow \overline{Me}$$

$$\sigma \leftarrow \frac{\overline{R}}{d}$$

となる．ただし，$\overline{Me}$ は（2 章で述べたように，管理図では通常 20 個の）Me の平均値，$\overline{R}$ は（2 章で述べたように，管理図では通常 20 個の）R の平均値，d は群の大きさ n によって一意に定まる値である．

よって，メディアン管理図の管理限界線は

$$UCL,\ LCL = \overline{Me} \pm 3\frac{m\overline{R}}{d\sqrt{n}} = \overline{Me} \pm A_2\overline{R}$$

となる．ただし，群の大きさ n によって一意に定まる値を $3\,m/d\sqrt{n} = A_2$ とした．

(平成 22 年 1 月 26 日受付)
(平成 22 年 5 月 26 日採録)

岩田　　聡（学生会員）

1983 年生．2007 年慶應義塾大学理工学部情報工学科卒業．2009 年同大学大学院理工学研究科開放環境科学専攻前期博士課程修了．現在，同専攻後期博士課程在学中．ディペンダブルコンピューティングの研究に従事．ACM 学生会員．

河野　健二（正会員）

1970 年生．1993 年東京大学理学部情報科学科卒業．1997 年東京大学大学院理学系研究科情報科学専攻博士課程中退，同専攻助手に就任．電気通信大学情報工学科講師等を経て，2005 年より慶應義塾大学理工学部情報工学科准教授．博士（理学）．オペレーティングシステム，分散システム，プログラミング言語システム等のシステムソフトウェア一般に興味を持つ．IEEE/CS，ACM，USENIX 各会員．