修 士 論 文

障害予測における最適な障害回避手段の提示法に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

加藤 裕

2013 年 3 月

修 士 論 文

障害予測における最適な障害回避手段の提示法に関する研究

指導教官　敷田幹文 教授

審査委員主査　敷田幹文 教授
審査委員　篠田陽一 教授
審査委員　知念賢一 特任准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1110020 加藤 裕

提出年月: 2013 年 2 月

概 要

本稿は，情報システムの可用性を維持・向上するためのアプローチとして障害予測技術に着目し，予測された障害からそれを回避する最適手段の自動的な生成・提示を行う仕組みを提案するものである．情報システムの大規模化や複雑化が進むなか，障害発生後の早期復旧だけでなく，障害の兆候を事前に検出し回避する運用管理手法への期待が増大すると考えられるが，障害の回避には高度な運用管理のスキルや経験が必要である事が多い．そこで，システムの大規模化や複雑化に対応しうる自動的な障害回避策の提示法を提案し，障害回避の機会を増やして障害そのものを減らす事で，情報システムの可用性の向上に繋げるのが本研究の狙いである．

目 次

第 1 章　はじめに **1**
1.1　研究背景 1
1.2　研究目的 1
1.3　本論文の構成 2

第 2 章　関連製品・研究 **3**
2.1　既存の運用管理製品 3
2.2　障害予測技術 3
2.3　運用管理製品の最新動向 4

第 3 章　推論システムを用いた予備実験 **5**
3.1　予備実験の概要 5
3.2　推論システムの検討 5
3.3　既知事実群の構造と表現 6
3.4　知識ベースの構造と表現 8
3.5　解決すべき諸問題 9

第 4 章　最適な障害回避手段の提示法 **10**
4.1　提案手法の概要 10
4.2　ルール生成自動化 10
4.2.1　概要 10
4.2.2　ルール詳細化アルゴリズム 12
4.2.3　詳細ルールによる回避策推論 13
4.3　最適回避策提示法 14
4.3.1　概要 14
4.3.2　最適回避策提示アルゴリズム 14
4.3.3　選択段階 15
4.3.4　評価段階 17
4.3.5　提示段階 18

第 5 章　評価実験 **19**
5.1　評価実験の概要 19

5.1.1 評価システムの実装 19
5.1.2 想定する運用管理構成と規模 20
5.1.3 入力データの記述 21
5.2 記述ルール数削減効果 25
5.3 最適回避策提示法における効果 25
5.4 性能評価 27
5.4.1 各処理ステップの実行時間の比較 27
5.4.2 入力したルール毎の実行時間の比較 27
5.4.3 ルールと処理ステップの関係 28

第 6 章 考察 **30**
6.1 大規模化への対応 30
6.2 複雑化への対応 31
6.3 熟練者への依存脱却 34

第 7 章 おわりに **35**
7.1 まとめ 35
7.2 今後の課題 36

謝辞 **37**

研究業績 **38**

参考文献 **40**

付 録 A ルール詳細化アルゴリズム記述 **41**

付 録 B 最適策提示アルゴリズム記述 **44**

図 目 次

3.1 回避策推論システムの構成 6
3.2 既知事実群 (システム構成・状態データ) の記述例 7
3.3 知識ベース (提案ルール) の記述例 8

4.1 ルール生成自動化の概要 10
4.2 ルール詳細化の動作例 11
4.3 ルール詳細化アルゴリズムで用いるデータ構造 12
4.4 ルール詳細化アルゴリズムを用いた回避策推論 13
4.5 最適回避策提示法の概要 14
4.6 最適回避策提示アルゴリズムの動作例 15
4.7 回避策候補における判断指標の情報収集・照合 16

5.1 評価システムの全体構成及び実行環境 19
5.2 詳細ルール生成数 25
5.3 選択段階の前後の回避策候補数 26
5.4 各処理ステップの実行時間の比較 27
5.5 各ルールの実行時間の比較 28
5.6 ルール毎の各処理ステップの実行時間の比較 29

6.1 ルール分割を用いた高複雑度の解消 32

表 目 次

4.1 選択段階における回避策選出のイメージ 17

5.1 評価実験の実施環境 20
5.2 主な構成要素名一覧 21
5.3 入力する基本ルール一覧 23
5.4 入力する実施コスト情報 24
5.5 基準値及び重み付け係数 24
5.6 評価値算出結果 26

6.1 各ルールのステップ毎の実行時間の比較 31
6.2 基本ルール 3 と 4 を含めた場合と含めない場合のステップ毎の実行時間の比較 31
6.3 想定する入力ミスとその対処 34

コードリスト 目 次

5.1 入力する障害予測情報 22
A.1 ルール詳細化アルゴリズム記述 41
B.1 最適策提示アルゴリズム記述 44

第1章　はじめに

1.1　研究背景

情報化社会が発展するにつれ，情報システムに要求される役割は日々増大している．今日の情報システムは，利用者数の増加，扱うデータの大容量化，サービスの 24 時間 365 日提供など様々な要求に答えながら，サービスレベルの維持・向上を目指して運用を行っている．一方で，情報システムが及ぼす障害の影響もこれまで以上に大きくなっており，銀行，証券取引所，交通機関，情報通信網などにおける情報システムの障害は我々の社会生活の妨げとなっている．加えて，クラウドコンピューティングの普及に伴い，1 つの障害が多数のサービスへ同時に影響を及ぼす事態も発生している．よって，情報システムの可用性向上が不可欠であるといえる．

情報システムの可用性 (Availability) は，MTBF (平均障害間隔) と MTTR (平均修復時間) の式によって表される．

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

このように，可用性を向上するには，MTBF を向上する施策と，MTTR を削減する施策の 2 通りがあることがわかる．運用管理においては，サービスの機能的・非機能的要件を満足できなくなることが障害であり，障害を未然に防ぐことが MTBF 向上に，障害から早期に復旧することが MTTR 削減に繋がる．特に MTTR 削減は，障害検出技術，根本原因解析技術，そして障害復旧手順の最適化など，多くの施策が研究され，また製品化されてきた．

一方で，情報システムの障害が引き起こす社会的影響が増大している今，これまでの高信頼なシステム設計や実装といった大規模障害を防ぐための様々な施策に加え，運用管理による MTBF 支援も求められるようになると考えられる．障害の予兆を事前に発見する障害予測技術を用いると，サービスダウンに繋がる障害をいち早く見つけ，回避することが可能になる．ただし，運用管理者が実際に障害を回避するには，システム構成や現在状態などを把握する能力など，高度なスキルや経験が求められる．

1.2　研究目的

本研究の主な目的は，幅広い知識と豊富な経験を持つ熟練の運用管理者へ依存せずに，障害予測から回避を実現するための支援を提供することである．しかし，情報システムは

大規模化の一途を辿っているほか，情報システムに要求されるニーズも多様化し，運用ポリシーは複雑化している．そこで，そのような状況下に置かれても障害の発生が迫る運用管理者へ迅速かつ適切な支援が可能な仕組みとして，知能情報処理の技術を中心にした障害回避策の推論と提示を行うシステムを提案することで，この問題の解決が可能かを明らかにする．評価においては，大規模な構成に複雑な運用ポリシーを組み合わせた環境を想定した実験を実施，複数の観点から手法を分析することで有用性を明らかにしたい．

1.3 本論文の構成

以下，第 2 章で運用管理に関する製品や研究について述べ，第 3 章で推論システムを用いた予備実験を行い問題点を議論する．その結果をふまえた上で，第 4 章で障害の最適回避策の提示法の提案を行う．第 5 章では，提案手法を実装した評価システムを用いて評価実験を行い，第 6 章で実験結果をふまえた考察を述べる．最後に，第 7 章でまとめと今後の課題を述べる．

なお，本文中では提案手法のアルゴリズムの詳細な記述は割愛した．代わりに付録としてそれらを末尾に掲載したので，併せて参照されたい．

第2章　関連製品・研究

本章では，既存の運用管理製品や，障害予測における研究について述べる．

2.1　既存の運用管理製品

運用管理を支援するソフトウェア等は数多く存在するが，特に有名な製品として HP OpenView や IBM Tivoli[1] が挙げられる．これらは以下のような機能を有している．

- サーバ管理機能
- ネットワーク管理機能
- 状態監視
- 障害分析
- ジョブ管理
- ワークフロー管理

これらの機能を有した製品は統合運用管理とも呼ばれ，日本国内の複数のベンダーも開発している．ある程度の規模の情報システムであれば，運用において必要不可欠なツールであり，幅広く利用されている．

しかし，今日の情報システムにおける非機能要件の複雑化，運用ポリシーの複雑化などに柔軟に対応しきれているとは言えず，運用管理者のスキル・経験で補われている側面が否定できない．また，可用性向上に注目すると，ダウンタイム削減 (MTTR 削減) に重点が置かれており，運用における MTBF 向上には乏しい．

2.2　障害予測技術

運用における MTBF 向上の施策として，障害を予測し回避する技術が挙げられる．今日までの障害予測技術は発展途上であるが，様々なアプローチで研究がなされている．

Sahoo ら [2] は，クラスタリングシステムに対し，複数の確率モデル的手法を用いてイベントを予測する研究を行った．また 2005 年には Bodik ら [3] が統計的手法を用い，予

測に繋がる障害の分析手法を提案した．後者は実際の Web アプリケーションサーバのログに対して実施されており，より実用的な評価がなされている．予測可能な障害の種類としては，サーバダウンに繋がるリソースの問題やパフォーマンスの問題などがあり，事前回避可能であれば可用性向上に寄与すると考えられる．

2.3 運用管理製品の最新動向

2012 年 3 月に，HP は障害予測技術を統合した新しい運用管理製品 HP Service Intelligence を発表した [4]．ハードウェア・ソフトウェアにおける様々な障害を過去の障害発生パターンと照合する等の複数の独自手法により予測し，運用管理者へ通知することができる．ただし MTBF 向上という観点からは，予測された障害の回避判断・措置は引き続き運用管理者に任されており，高いスキル・経験を持つ運用管理者が必要である．

2012 年 4 月には，IBM がエキスパート (専門家) の知見をパターン化し，顧客に合わせてそれらのパターンを組み合わせて最適な運用管理を提供する PureSystems を発表した [5]．運用管理者の負担削減という意味で価値があるが，専門家の知見を顧客のニーズに摺り合わせるのは IBM のエンジニアによる人手であり，熟練技術者からの真の依存脱却という課題が残っていると言える．

第3章　推論システムを用いた予備実験

本章では，提案手法の議論に先立って実施した予備実験について述べる．

3.1　予備実験の概要

障害を回避するには，対象のシステムに関する情報や障害予測情報，過去の事例などを総合的に判断した上で適切な障害回避策を下す必要がある．しかし，熟練の運用管理者に依存せずにこれを可能にするにはこれら高度な判断を自動化する必要がある．そこで，熟練者の知識・判断能力を再現するために，知識情報処理における既存の手法を用いて問題を分析する．

3.2　推論システムの検討

1970 年代より，状態に応じて推論を行うシステムやモデルは複数考案されている．代表的なものに，エキスパートシステム [6]，事例ベース推論 [7]，ベイジアンネットワーク [8]，そして様々な機械学習のアルゴリズムを用いる手法などが挙げられるが，今回はエキスパートシステムを用いて作成した．

エキスパートシステムとは，既知事実群 (Facts) に対し知識ベースとなるルール (Rules) を連鎖的に適用させることで，新たな事実群を得るシステムである．今回，回避策を推論するにあたりエキスパートシステムを採用した理由には，主に以下のような特徴を有している事があげられる．

1. 既知事実群・知識ベースとも自然言語 (文章) で記述でき，扱いやすい．
2. 全ての推論結果が論理的に正しい事が保証されている．
3. 複数の入力に対し，状況に合致する複数 (全て) の結果を得られる．

エキスパートシステムの欠点として「ルールに書かれていないことは答えられない」という問題点がよく議論される．例えば，事例ベース推論では知識を修正しながら近似解を導出する性質を有しており，エキスパートシステムの欠点を解決している．しかし，今回の推論で導出したい障害予測における障害回避手段は，障害発生前の稼働中のシステムに対して施す手段を導出するものであるため，近似的な解 (論理的な保証がない解) を試行

する事によって状況を悪化させる事は絶対に避けなければならないという制約下にある．そこで今回はこの欠点を逆に取り，利点として挙げて採用した．

予備実験で実施する回避策推論システムの全体構成を図 3.1 に示す．

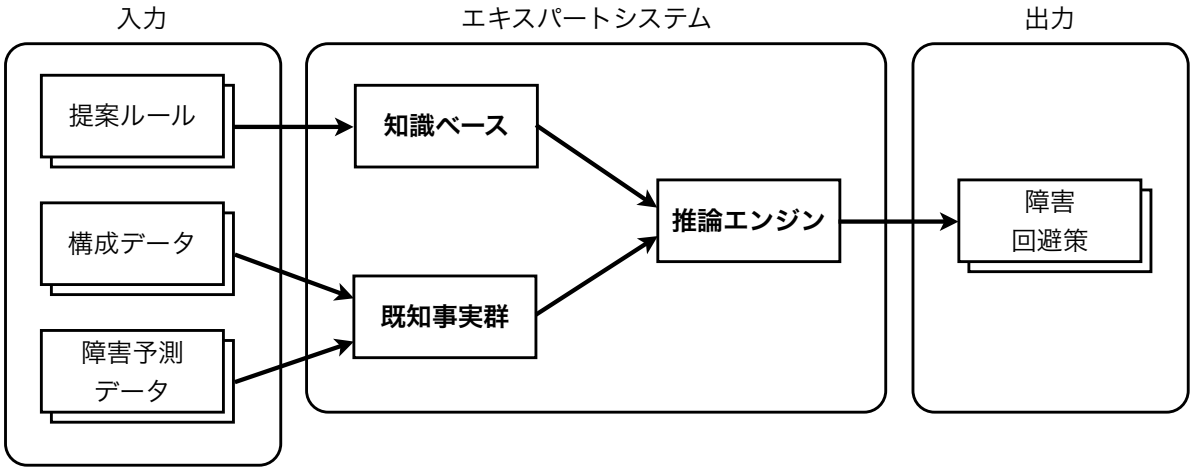

図 3.1: 回避策推論システムの構成

既知事実群に情報システム構成や障害予測に関する情報を，知識ベースに回避判断を集めた運用ルールを入力することで回避策を得る仕組みである．なお，実装は Java SE 7[9] を用いて記述した．

3.3 既知事実群の構造と表現

既知事実群は事実の集合であり，対象システムの次の情報が含まれる．

- システムの静的な情報
  - 各構成要素の詳細情報 (性能など)
  - 構成要素間のネットワークに関する情報
  - 各構成要素が提供するサービスの情報
- システムの動的な情報
  - 構成要素間の依存関係
  - 各構成要素の動作状態
  - リソース状態
- 障害予測情報

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE facts[
        <!ELEMENT facts (fact+)>
        <!ELEMENT fact (#PCDATA)>
]>
<facts>
        <fact>HOST-FS1 は稼働中</fact>
        <fact>HOST-FS2 は停止中</fact>
        <fact>HOST-Web1 は稼働中</fact>
        <fact>HOST-Mail1 は稼働中</fact>
        <fact>HOST-FS1 は SERVICE-FS を提供中</fact>
        <fact>HOST-Web1 は SERVICE-Web を提供中</fact>
        <fact>HOST-Mail1 は SERVICE-Mail を提供中</fact>
        <fact>HOST-FS1 は HOST-SW1 に接続中</fact>
        <fact>HOST-FS2 は HOST-SW1 に接続中</fact>
        ・・・
</facts>
```

図 3.2: 既知事実群 (システム構成・状態データ) の記述例

今回実装する上でこれらの情報の一部を表現した例を図 3.2 に示す.

各事実は自然言語で記述し，これらが後述する提案ルールを適用するための条件になる.

障害予測情報に関しては，用いる障害予測技術によって大きく内容が変わり，提案ルールの策定・適用にも大きな影響を与える事が考えられる．既存の障害予測技術である Web アプリケーションにおけるアクセス時間解析方式による障害予測技術 [10] の研究では，障害予測技術の各障害予測の出力として次の情報が得られる事が読み取れる.

- 異常度 (障害予測の疑わしさ)
- 障害予測の発生原因場所

また，複数の障害予測の結果を分析する事により，次の情報も得られる.

- 障害予測の精度
- 障害予測の発生原因場所特定の精度
- 障害発生の大まかな予想時刻

本システムでは，これらを障害予測発生時に収集・既知事実群へ追加する事を想定している.

3.4 知識ベースの構造と表現

知識ベースは IF-THEN 形式のルールの集合であり，今回実装する上で提案ルールを表現した例を図 3.3 に示す.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rules[
        <!ELEMENT rules (rule+)>
        <!ELEMENT rule (if+, then+)>
        <!ATTLIST rule name CDATA #REQUIRED>
        <!ELEMENT if (#PCDATA)>
        <!ELEMENT then (#PCDATA)>
]>
<rules>
        <rule name="FS切換ルール1">
                <if>HOST-FS1 に障害予測が発生</if>
                <if>HOST-FS1 は SERVICE-FS を提供中</if>
                <if>HOST-FS2 は SERVICE-FS を提供可能</if>
                <if>HOST-FS2 は 停止中</if>
                <then>HOST-FS2 を起動する</then>
                <then>HOST-FS2 へ切替える</then>
        </rule>
        <rule name="FS切換ルール2">
                ・・・
        </rule>
        ・・・
</rules>
```

図 3.3: 知識ベース (提案ルール) の記述例

内容である提案ルールは，既知事実群である対象システムの構成・状態及び障害予測情報に基づいたものである．各ルールの if 要素が推論エンジンによって既知事実群と照合され，条件に合う then 要素が新たな既知事実群に追加される．推論エンジンは新たな既知事実が追加されなくなるまでこれを繰り返す事で，最終的な推論を導く事ができる．

知識ベースは自由にルールを増減させる事ができる拡張性，自然言語が利用できる事による可読性に加え，複数のルールや条件を組み合わせた複雑な運用ポリシーが容易に記述可能である．記述例では XML をベースとした記法を採用しているが，この場合 1 つの rule 要素中に複数ある if 要素や then 要素は論理積に相当し，複数の rule 要素に同一の

if 要素がある場合は論理和に相当する．これによって，複雑な条件式でもルールを単位とした柔軟な制御が可能である．

3.5 解決すべき諸問題

20 件の既知事実群と提案ルールを手動で作成し，今回実装した回避策推論システムに入力したところ，事実に適合する 3 件の障害回避策が特に問題なく出力され，本予備実験によって自動化できる事を確認した．そこで，この回避策推論システムが研究目的を達成するために充分であるかについて議論を行った．

議論の結果，懸念された問題点は以下の 3 点である．

問題 1) ルール数爆発 システムが大規模化した場合に，提案ルールの管理が追いつかなくなる可能性

問題 2) 判断指標考慮 提示する回避策は様々な判断指標を考慮しなければならない

問題 3) 回避策安全性 誤ったルールや他のシステムのルールが混ざった場合，正しい回避策を提示できない可能性

ルール数爆発については，システムの規模によっては提案ルールの数が膨大になり運用管理者の手に負えなくなる問題や，構成変化に応じてルールの策定し直しが必要になるといった問題を指す．熟練でない運用管理者が扱えるようにする為にも，こうした問題は解決しなければならない．

問題 2 については，熟練の運用管理者が実施する回避策の判断を再現するために，システムが最適な回避策を提示する必要があるという事である．回避策推論システムのみでは実施可能な回避策が複数提示される可能性があり，その場合それらの優劣は運用管理者が評価し最終的に 1 つの回避策を決定する必要がある．判断指標とは，時間的・金銭的コストやサービスレベルといった指標で，最善の回避策を選択する上で必要になる．また，障害回避においては対象となる障害はその時点では発生していないため，障害発生予想時間や予測精度も考慮した上で回避策を選択すべきである．

問題 3 については， 熟練でない運用管理者が扱うシステムを想定している以上，ルール記述に誤りがあった場合でもシステムを誤った回避策の実行から守る必要がある事と，ルールをテンプレート化して共有することで，他のシステム向けのルールに従った回避策が提示される可能性があるという事である．なお，ルール記述法の誤りは文書型定義[1] により事前に確認されるほか，論理的に矛盾するルールはエキスパートシステムで採用されずに回避できる．しかし，システムに適合しないルールがエキスパートシステムにより採用され提示されてしまう可能性は残るため，これを排除しなければならない．

[1] XML DTD (Document Type Definision) や XML Schema によって実行する．

第4章　最適な障害回避手段の提示法

本章では，提案手法である最適回避策の提示法について述べる．

4.1　提案手法の概要

第 3 章の予備実験では，ルール数爆発，判断指標考慮，回避策安全性の 3 つの問題を提起した．そこで，それらを解決し本研究の目的を達成するために，以下の 2 つの手法を提案する．

- ルール生成自動化
- 最適回避策提示法

以下，順に説明する．

4.2　ルール生成自動化

4.2.1　概要

前述の回避策推論の問題 1 を解決するため，提案ルールにおいて実効的なルールを最初から全て用意するのではなく，必要最低限のルールから不足している情報やルールを自動的に展開し統合する仕組みを提案する．概要を図 4.1 に示す．

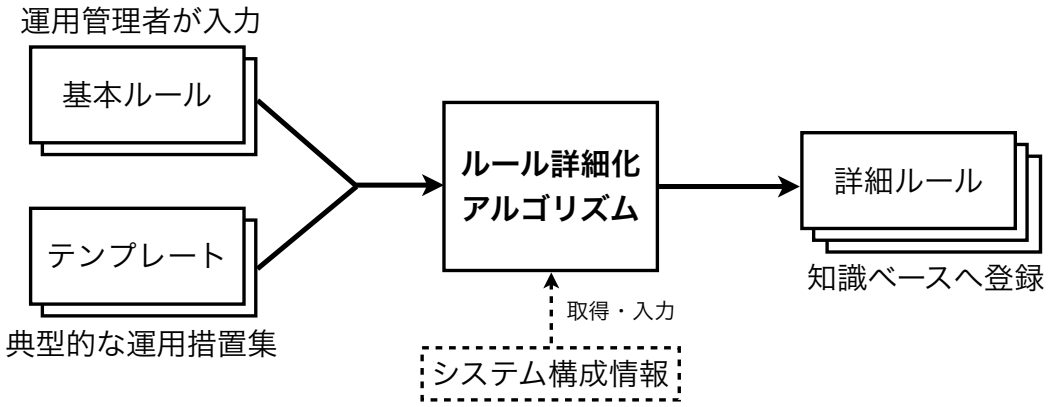

図 4.1: ルール生成自動化の概要

基本ルールは，運用管理者が対象システムの運用方針に基づき必要最小限の情報を入力する．構成要素 1 台 1 台に対する記述を避け，スケールアップする部分を抽象化することで，基本ルールでは回避策の大まかな記述で済む．更に，あらゆるシステムに適用可能な様々な運用措置をテンプレートとしてまとめることで，システム間で流用・再利用を可能にする．

具体的なルール詳細化の例を図 4.2 に示す．

図 4.2: ルール詳細化の動作例

基本ルールやテンプレートでは適用対象は決定されていない．このようにする事で，運用管理者はシステムの詳細な構成を把握する必要がなく，また変更点がある場合にメンテナンスが困難になる可能性を排除する．詳細化処理では，ルール中の曖昧性を自動的に収集した構成情報を基に照合し，必要であればルールを場合分けする．これにより，最終的に既知事実群に適用可能な実効的な詳細ルールが生成される．

4.2.2 ルール詳細化アルゴリズム

これらの機能を実現するために，ルール詳細化アルゴリズム (付録 A コードリスト A.1) を考案した．入力された抽象的なルールに対し，自動的，反自動的に取得した構成情報を用いることで，実施可能な全ての振る舞いを導出する．

図 4.3 は，本アルゴリズムで振る舞いの全パターン，組合わせを流し込む中間データの構造である．

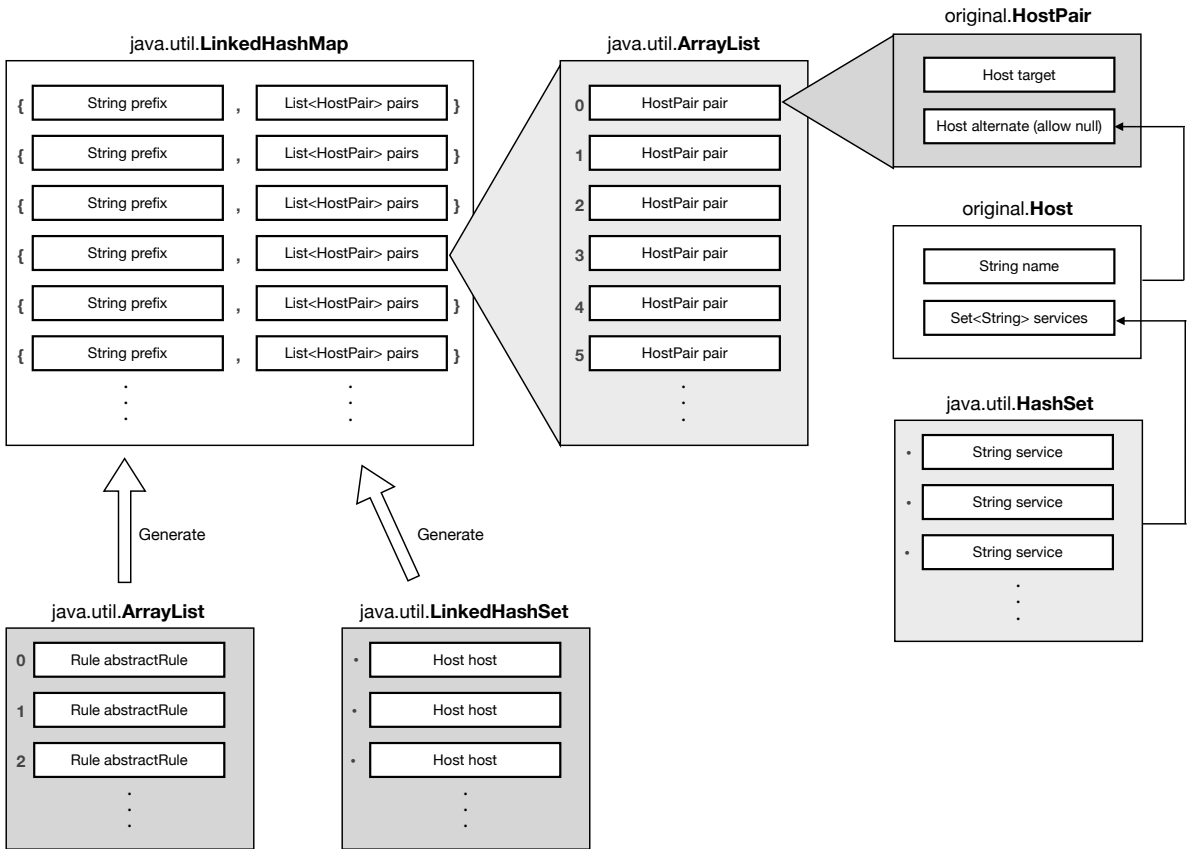

図 4.3: ルール詳細化アルゴリズムで用いるデータ構造

本アルゴリズムは，引数で受け取ったルールと構成情報を基に，まず上記データ構造に詳細化パターンを構築する．次に，このパターンをループを繰り返しながら引数で受け取ったルールに適用してゆき，詳細ルールを出力するというものである．ルール中で “他の xx” という記述が出現した場合は，全ての切換の組合わせを生成するためのループに入り，上記データ構造の HostPair クラスの一覧で切換の組合わせを表現する．逆に切換えがない場合は，切換先欄を null の状態で後の処理に引き渡される．

4.2.3 詳細ルールによる回避策推論

詳細化アルゴリズムに基本ルールを入力し，実際に障害予測が発生したときに最終的にどのような出力になるかを示したものが図 4.4 である．なお，ここまでは結果に影響を与える部分のみ図示しているが，実際にはシステムの規模に応じた多数の既知事実群の中から状況に適した回避策を選択・提示する．

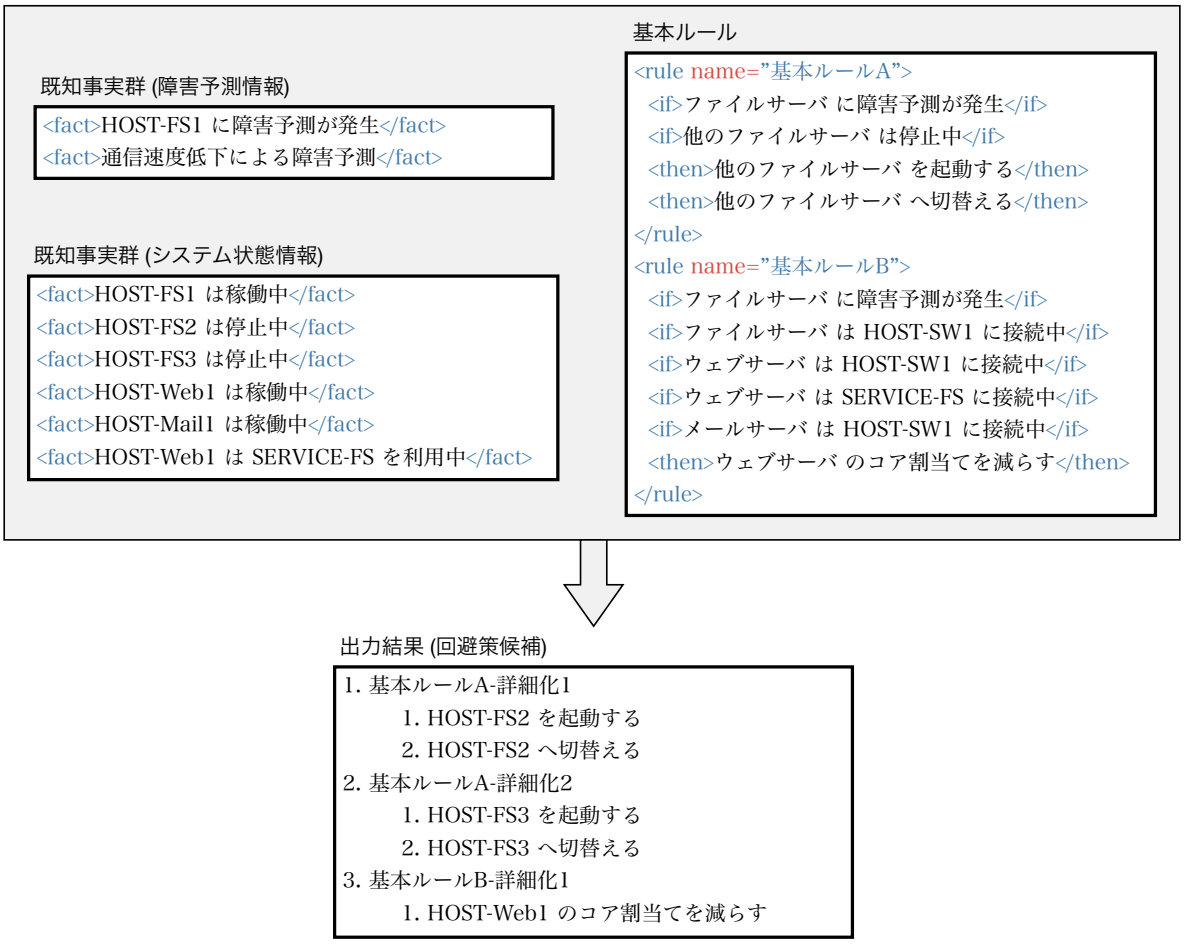

図 4.4: ルール詳細化アルゴリズムを用いた回避策推論

障害予測が発生した時点で，既知事実群に障害予測に関する情報，及び発生時点でのシステムの動的な情報が追加される．全ての既知事実群がそろったところで回避策推論システムにそれらが入力され，詳細ルールの適用を行う．回避策推論システムの出力は，推論エンジンによって新たに追加された既知事実の集合であり，これが現状に即した有効な回避策の候補という事になる．

4.3 最適回避策提示法

4.3.1 概要

前述の問題 2 は，提示する回避策は様々な判断指標を考慮しなければならないというものであった．回避策推論システムが出力する結果は、どれも障害回避に繋がると推論された有効なものだが，この中には SLA[1] に不適合な回避策や，障害予測における回避可能時間内に達成できないと見込まれる回避策などが含まれる可能性がある．さらに，熟練でない運用管理者でも扱うことを可能とするには，有効な回避策の候補から実際にどの回避策を実行に移すのが適切かを判断するための支援も必要である．

同問題 3 は，誤ったルールや他のシステムのルールが混ざった場合を考慮しなければならないというものであった．誤ったルールによってシステムに効果のない，あるいは悪影響を与える回避策を提示してしまう可能性を排除するためにも，同様に回避策候補の適・不適を判断する必要がある．

そこで，安全で最適な回避策を提示するための仕組みを提案する．概要を図 4.5 に示す．

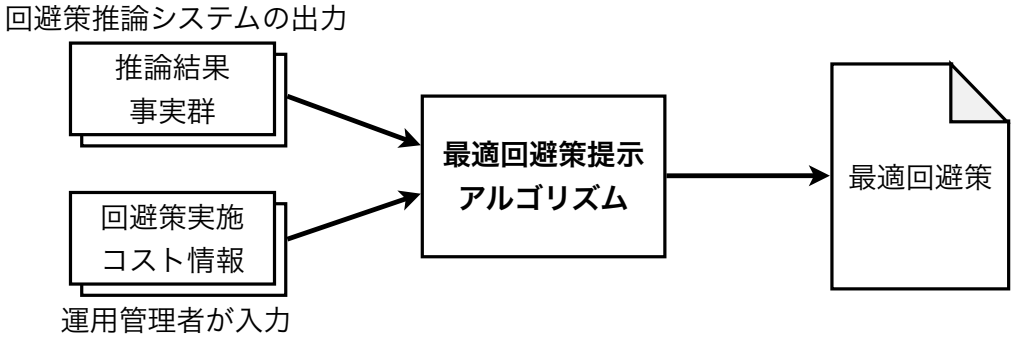

図 4.5: 最適回避策提示法の概要

回避策推論システムの出力を詳細に分析し，最適回避策を導出し，最終的に運用管理者へ提示するものである．以下アルゴリズムの内容について述べる．

4.3.2 最適回避策提示アルゴリズム

最適回避策提示アルゴリズムの内容は，次の 3 つの段階に大別できる．

選択段階 回避策推論結果のうち現状に適合しない回避策を除外する．

評価段階 適合する各回避策について評価値を算出する．

[1]Service Level Agreement

提示段階 最も評価値が高い回避策を最適策，その他を代替策として提示する．

具体的に，複数の回避策推論結果から最適策が決定し提示されるまでの例を図 4.6 に示す．

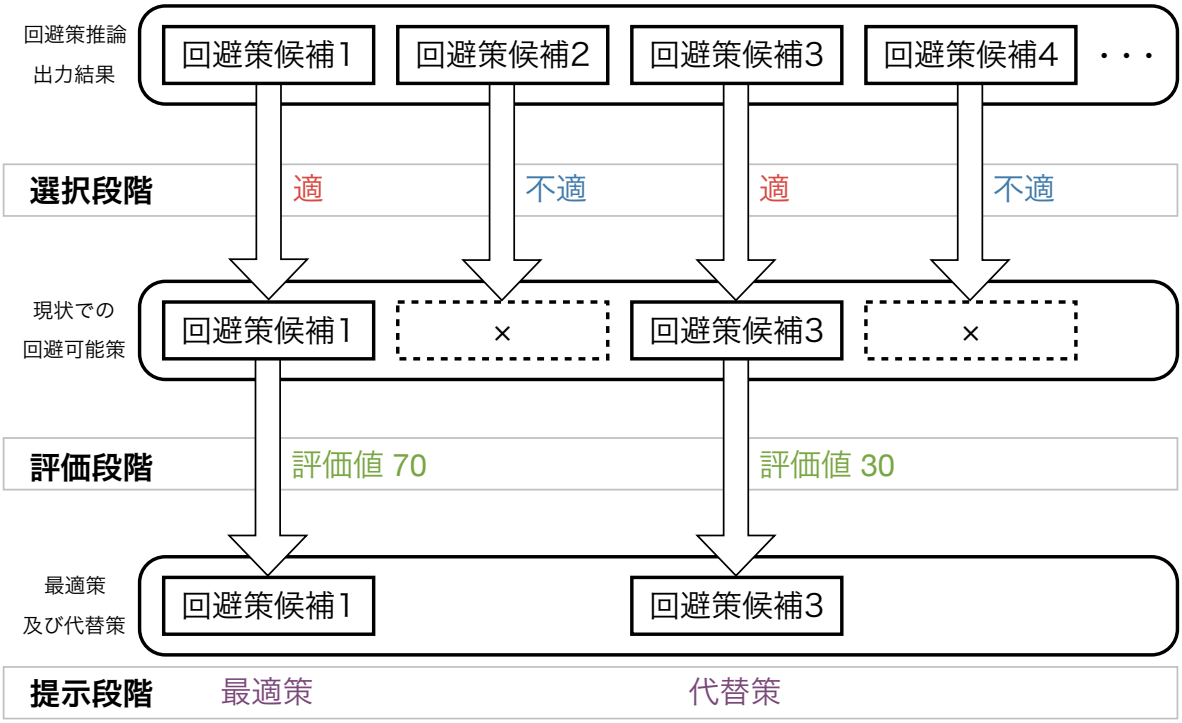

図 4.6: 最適回避策提示アルゴリズムの動作例

以下，各段階ついて順に説明する．

4.3.3 選択段階

まず，各回避策の評価項目の情報収集を行い結果を照合する．前小節で示した出力例を用い，最適策を判断するための必要な情報の収集と照合の様子を示したのが図 4.7 である．

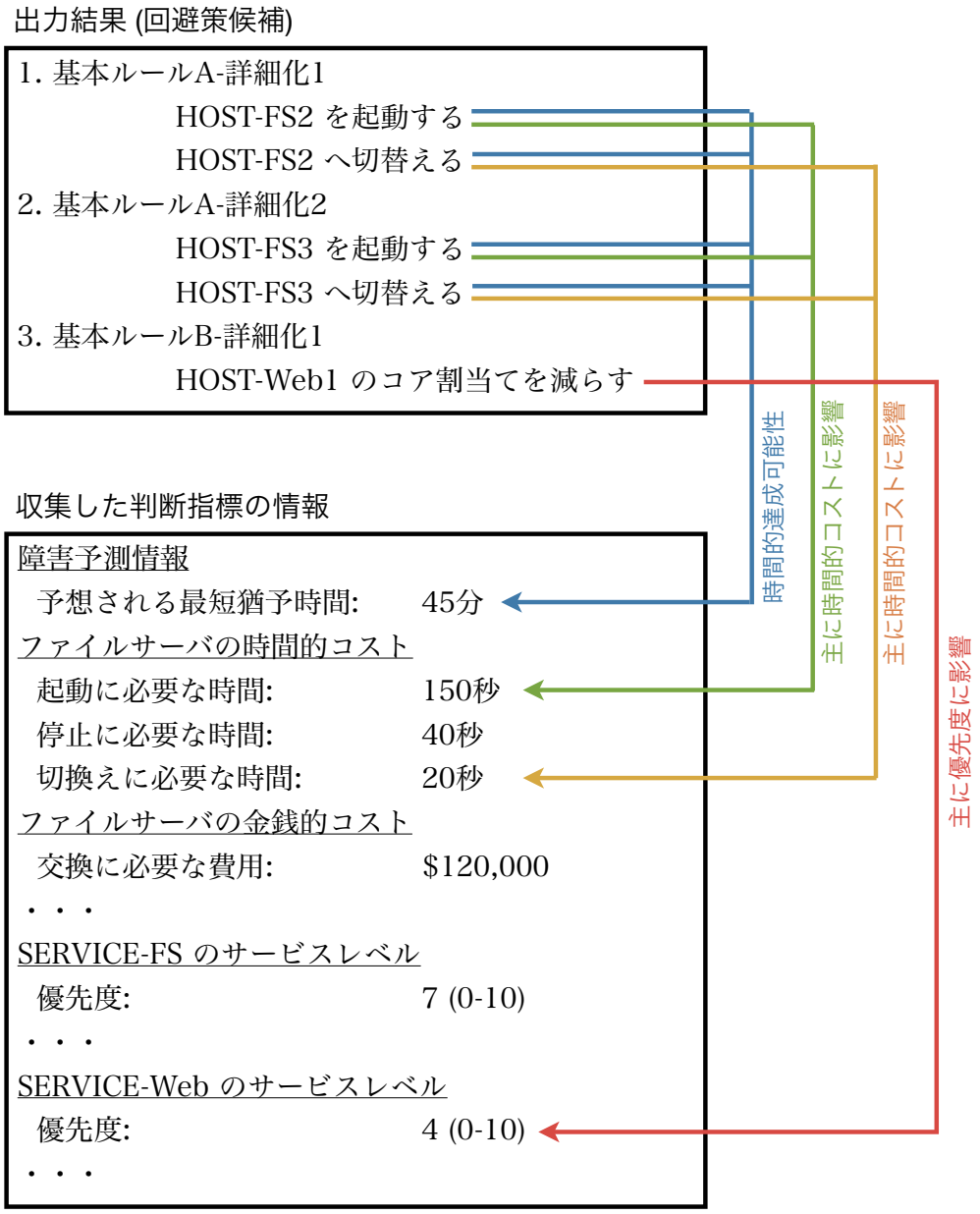

図 4.7: 回避策候補における判断指標の情報収集・照合

各構成要素の時間的コスト，金銭的コスト，サービスレベルは予め対象システムを設計した者が与える．また，障害予測情報は発生時点で得られたものである．

次に，これらを表にまとめ，基準に基づいて適・不適を判断する．各回避策候補と収集した判断指標から最適回避策を選出する様子の例を表 4.1 に示す．

表 4.1: 選択段階における回避策選出のイメージ

回避策	実施時間	費用	SLA	時間内	結果
回避策 1	30 分	¥150,000	OK	OK	適
回避策 2	90 分	¥50,000	OK	NG	不適
回避策 3	40 分	¥300,000	OK	OK	適
回避策 4	2 分	¥0	NG	OK	不適

ここで言う基準とは，各評価項目における次のようなものを指す.

- 予想される最短猶予時間内に回避策を実行できるか.
- 予め設定した金銭的コストの限度.
- サービスレベルの各基準，SLA.

基準に合わない回避策候補は不適と判断され，最終的な回避策の提示には反映しない.

4.3.4 評価段階

基準に合わない回避策を消去したとしても，その中からどれが現状に最適な回避策なのか判断するのは容易ではない．そこで，熟練でない運用管理者でも最終的な障害回避策 (1 つ) を決定できるよう支援するため，評価値を算出してランキングを行う.

回避策候補 x の評価値 $Score$ の算出方法を次に示す．ここで，t を各コストの種類 (時間的コスト，金銭的コスト，サービス優先度など)，T をその集合 ，C をコストの値，B をコストの基準値 (適・不適を判断する上の境界となる値)，および W をコストの重みとおく.

$$Score_x = \sum_{t \in T} W_t(C_t x - B_t x)$$

これにより最適回避策は，最適策評価値を $Best$ とすると，

$$Best = max(Score_x)$$

となる回避策候補 x である.

考慮すべき点として，コストの種類ごとの重み W_t をどのように設定するかで最適策が変動する点が挙げられる．標準的な運用では， $W_t(B_t)$ が t ごとに一様になるような W_t を設定する事を想定しているが，例えばサービス優先度 $priority \in T$ をより強く反映したい場合は $W_{priority}$ を引き上げて調整するといった措置が考えられる.

4.3.5 提示段階

提案手法の出力として運用管理者へ最適策のみを提示するという方式も考えられるが，本研究で目指しているのは熟練でない運用管理者の支援であり，自律運用ではない．そこで最終的な運用管理者への提示は複数の回避策を $Score_x$ で並び替え，各評価項目の評価結果も併せて提示する事を想定している．

回避策一覧に含まれる主な提示項目は，

- 回避策の内容 (実施手順)
- 評価値，評価値に最も影響を及ぼした評価項目[2]
- 実施時間，費用などの評価項目の内容

これにより，もし熟練でない運用管理者が近い $Score_x$ 同士の回避策で迷った際には，他の運用管理者に相談して最終的な回避策を決定したり，またある程度スキルや経験を持った運用管理者が提案手法の出力を参考にしてより最適な回避策がないか検討したりといった運用支援が期待できる．

[2] $max(W_t(C_t x - B_t x))$ となるコストの種類 t を示す．

第5章　評価実験

本章では，第 4 章で述べた提案手法について，その実装を行い実施した評価実験について述べる．

5.1　評価実験の概要

提案する障害予測における最適な障害回避策の提示法を実装した評価システムを用い，提案手法の有用性を評価する実験を行った．

5.1.1　評価システムの実装

評価システムは，第 4 章の節 3.1 の予備実験で作成したプログラムを STEP2 に配置し，その前後に 2 つの提案手法を統合したものである．全体構成を図 5.1 に示す．

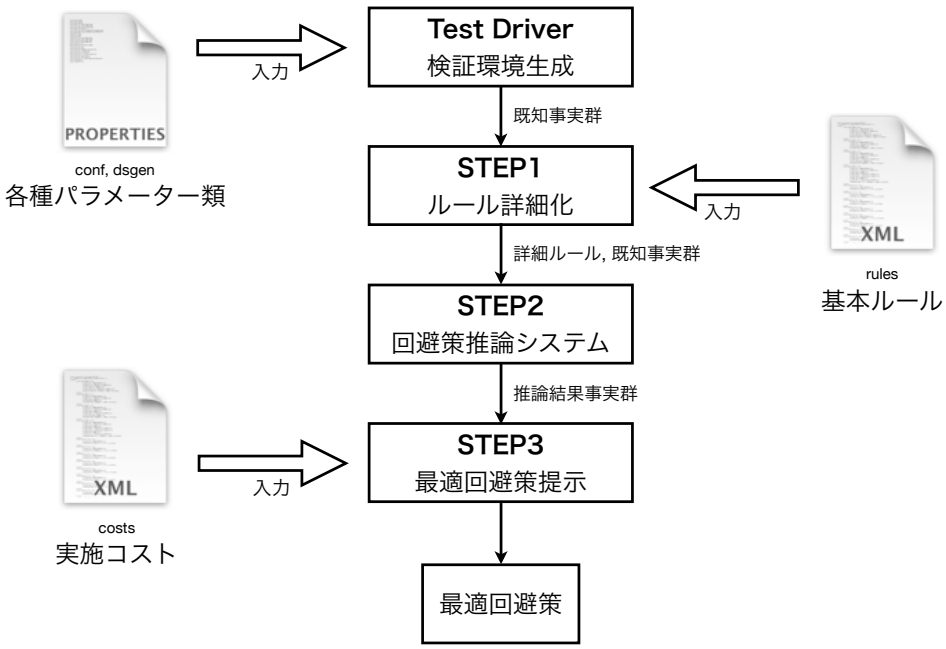

図 5.1: 評価システムの全体構成及び実行環境

プログラムは予備実験と同様，Java SE 7 [9] で実装し，また一部に関数型の機能を統合したオブジェクト指向言語である Scala 2.9 [11] を用いた．

なお図 5.1 中の Test Driver の役割は，提案手法の検証のために，必要な入力パラメーターを受取り，想定する環境を生成，そこへ各ステップを実行し，回避策と様々な統計データを取得する事である．また，今回は複数の入力パラメーターを用いて構成をスケールアップする評価実験を実施するために，それらを自動化する仕組みも実装した．これにより，STEP1 から最適回避策提示までの一連のプロセスを入力を変えながら自動的に試行する事を可能とした．

次に，評価システムの実行に用いたマシンの環境についてまとめたものを表 5.1 に示す．

表 5.1: 評価実験の実施環境

項目	内容
マシン	Apple Mac Pro Mid 2010
プロセッサ	3.33 GHz 6-Core Intel Xeon
メモリ	6 GB 1333 MHz DDR3 ECC
OS	Mac OS X 10.8.2 “Mountain Lion”
JRE	Oracle Java SE 7 update 11 64bit

中・大規模システムの運用管理の現場で用いられている管理用端末の性能を鑑み，高性能な並列計算機ではなく，身近にあるマシンの中でやや高性能なものを選択した．後述する性能評価の議論においては，ここで述べたマシン選択について特に考慮されたい．

なお， Java 仮想マシンのヒープサイズなどの設定はデフォルト [1] を用いている．Scala で実装した部分においても，バイトコード実行は Java 仮想マシン上なので同様である．

5.1.2 想定する運用管理構成と規模

評価実験を行うにあたって，以下のようなシステムを直接管轄する運用管理を想定した．

- 構成要素数 (サーバ台数) は数台から 100 台，200 台規模まで．
- 3 階層システム [2] でサービスを提供．
- 複数の構成要素で各層の負荷を分散して支える構成．
- 可用性，金銭的コスト，パフォーマンス，信頼性に厳格なサービスレベルを規定．

[1] 通常は物理メモリの 1/4 程が最大値に設定される．

[2] “プレゼンテーション層”, “アプリケーション層”, “データ層” から成るクライアント/サーバーシステム

- 稼働中の構成要素は他層のサービスを参照，依存関係にある．
- 各構成要素は，物理ホスト，仮想ホストどちらも想定する．

評価実験では，このようなシステムの運用管理者にとって，提案手法がある場合とない場合でどれほどの違いが想定されるかという観点で実施，議論を行う．

5.1.3 入力データの記述

節 5.1.1 全体構成で言及した通り，評価実験の入力データとしては主に以下の 3 種類がある．

- 既知事実群 (構成要素情報, 障害予測情報)
- 基本ルール
- 実施コスト情報

それぞれについて以下に詳しく説明する．

既知事実群

まず，各種構成要素や層 (サービス) に名前を付ける必要がある．表 5.2 に，今回ルール記述で用いた構成要素名及びサービス名を示す．

表 5.2: 主な構成要素名一覧

構成要素名の接頭辞	説明	機能
HOST-FS	データサーバ	サービス “SERVICE-FS” を提供
HOST-AP	アプリケーションサーバ	サービス “SERVICE-AP” を提供
HOST-Web	ウェブサーバ	サービス “SERVICE-Web” を提供
HOST-SW	スイッチ	接続関係表現用
HOST-RT	ルータ	同上

実際の構成要素を識別する際は，構成要素名の接頭辞の後ろに通し番号を添える[3]．評価実験では，構成要素数を Test Driver で増加させていく．規模 (Scale) を 1 とした場合は 3 つのサービスを提供する構成要素がそれぞれ 1 つづつ， 規模を 2 とした場合は 2 つ

[3]例) HOST-FS3 で認識順 3 番目のデータサーバを指す．

づつというように増加させる事とする．なお，本評価実験ではサービスを提供する構成要素に関する分析を主な対象とし，接続関係に関してはフラットな構成としている．

各構成要素名はここで挙げた以外にも既知事実群内で自由に定義可能であり，構成要素の種類を増やしたい場合は，既知事実群にその構成要素を記述し，対応するルールを用意する事で回避策推論に加えることができる．

障害予測情報については，評価実験では以下の情報を入力する．

コードリスト 5.1: 入力する障害予測情報

```
1 <facts>
2   <fact type="prediction">HOST–FS1 に障害予測が発生</fact>
3   <fact type="reason">処理能力不足による障害予測</fact>
4 </facts>
```

障害予測原因の記述もルールの条件に含めることで，より高度な判断を自動化できる．しかし，全ての障害予測技術がこれを正しく提供できるとは限らないので，注意が必要である．ここでは，あくまで表現力を説明するもので，評価実験で入力する基本ルールの各条件にはこれを含めていない．

基本ルール

提案手法の一つ，ルール詳細化 (評価システム STEP1) に入力する，実験用に記述した様々なルールについて説明する．今回分析システムに入力する基本ルールには，表 5.3 に示す通り性質が異なる以下の 4 つのルール (群) を用意した．

表 5.3: 入力する基本ルール一覧

ルール名 (省略名)	主な回避操作	構成要素種類数	切換
テンプレート 1 (T1)	「HOST-FS* を停止する」	1	-
テンプレート 2 (T2)	「HOST-FS* を再起動する」	1	-
テンプレート 3 (T3)	「HOST-FS* を交換する」	1	-
テンプレート 4 (T4)	「HOST-AP* を停止する」	1	-
テンプレート 5 (T5)	「HOST-AP* を再起動する」	1	-
テンプレート 6 (T6)	「HOST-AP* を交換する」	1	-
テンプレート 7 (T7)	「HOST-Web* を停止する」	1	-
テンプレート 8 (T8)	「HOST-Web* を再起動する」	1	-
テンプレート 9 (T9)	「HOST-Web* を交換する」	1	-
基本ルール 1 (B1)	「他の HOST-FS* へ切り替える」	1	有
基本ルール 2 (B2)	「HOST-Web* を再起動する」	2	-
基本ルール 3 (B3)	「HOST-Web* を再起動する」	2	有
基本ルール 4 (B4)	「HOST-Web* のコア割当てを減らす」	3	-

構成要素種類の後ろに付く“ * ”(アスタリスク) は，任意の構成要素一つを意味する抽象記述である．

表中の構成要素種類数とは，一つのルール中にいくつの構成要素の種類 (前述の HOST-FS, HOST-AP など) に言及があるかを示している．この数字が多いほどルール詳細化で生成されるルール数が増加する事になる．また，同じく表中の切換とは，ルール中に代替構成要素を探す記述が含まれるかを示している．評価システムでは，障害予測が発生した際に代替構成要素を探す記述を容易にルール化できるよう実装した．しかし，この記述を行うと組合わせが膨大になる事が考えられる．そこで，この機能の性能を検証するために用意した．

実施コスト情報

提案手法の一つ，最適回避策提示法 (評価システム STEP3) に必要な実施コスト情報について説明する．実施コストとは，ルールに記述された回避操作に必要なコスト (時間的コスト，金銭的コスト，サービスレベルなど) を意味する．評価実験で入力する実施コスト情報を表 5.4 に示す．

表 5.4: 入力する実施コスト情報

回避操作	該当ルール	時間	費用	性能	信頼度
「HOST-FS* を停止する」	T1	50.0	30	0.0	1.0
「HOST-FS* を再起動する」	T2, B2, B3	180.0	200	100.0	1.0
「HOST-FS* を交換する」	T3	1800.0	8000	100.0	1.0
「HOST-AP* を停止する」	T4	50.0	30	0.0	1.0
「HOST-AP* を再起動する」	T5	180.0	200	100.0	1.0
「HOST-AP* を交換する」	T6	1800.0	5000	100.0	1.0
「HOST-Web* を停止する」	T7	50.0	30	0.0	1.0
「HOST-Web* を再起動する」	T8	180.0	200	100.0	1.0
「HOST-Web* を交換する」	T9	1800.0	4000	100.0	1.0
「他の HOST-FS* へ切り替える」	B1	60.0	200	100.0	1.0
「HOST-Web* のコア割当てを減らす」	B4	30.0	100	70.0	1.0

各指標の単位は，時間 (sec.)，費用 ($)，性能 (%)，そして信頼度は比率である．より多くの評価指標を含めることも可能であるが，今回の評価実験においてはスコアリングの有用性を示すための必要最小限とした．

次に，実施コスト情報と併せて利用するパラメーターである，基準値及び重みづけ係数について説明する．それぞれまとめたものを表 5.5 に示す．

表 5.5: 基準値及び重み付け係数

項目	時間	費用	性能	信頼度
基準値	180.0	1000	0.1	0.0
重み付け係数	-0.44	-0.1	1.0	1.0

評価実験では，重み付け係数を最も望ましい値で 100，基準値で 0 となるように設定した．これらは設定ファイルのキーとして用意してあり，実行毎に変更可能である．評価実験では，全て上記の値に固定して実施する．

5.2 記述ルール数削減効果

まず，提案手法の一つであるルール詳細化によって，複数の基本ルールが存在する環境においてどの程度の詳細ルールが生成されるかを調査した．

用意した全てのルール (基本ルール 1 〜 4，及びテンプレート 1 〜 9) を同時に入力した際に算出した詳細ルール数の推移を図 5.4 に示す．横軸は構成要素数，縦軸はルール数を表す．

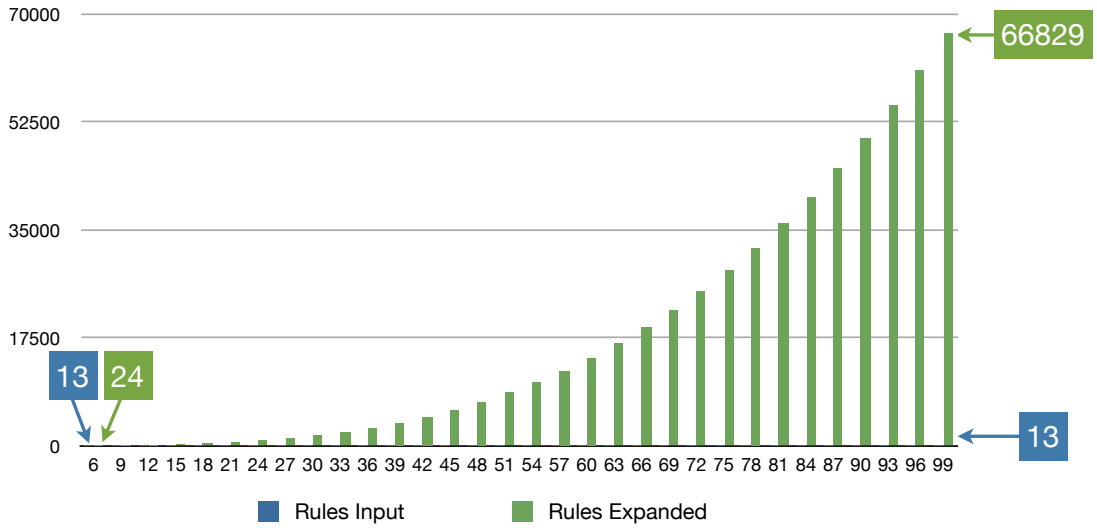

図 5.2: 詳細ルール生成数

入力した基本ルール 13 件に対し，構成要素数に応じてそれらの組合わせを展開することで，詳細ルールの生成数が増加し，構成要素数が 99 の場合には詳細ルールが 66829 件生成されている事がわかる．

運用を行う上で，全ての詳細ルールが実際に利用される訳ではないものの，同種の構成要素が多数配置される情報システムでは，その構成要素数が多ければ多いほど手間を省ける事になる．

5.3 最適回避策提示法における効果

用意した全てのルール (基本ルール 1 〜 4，及びテンプレート 1 〜 9) を同時に入力した際に算出した全回避策候補数と選択段階で選択された回避策候補数を図 5.3 に示す．横軸は構成要素数，縦軸は回避策候補数を表す．

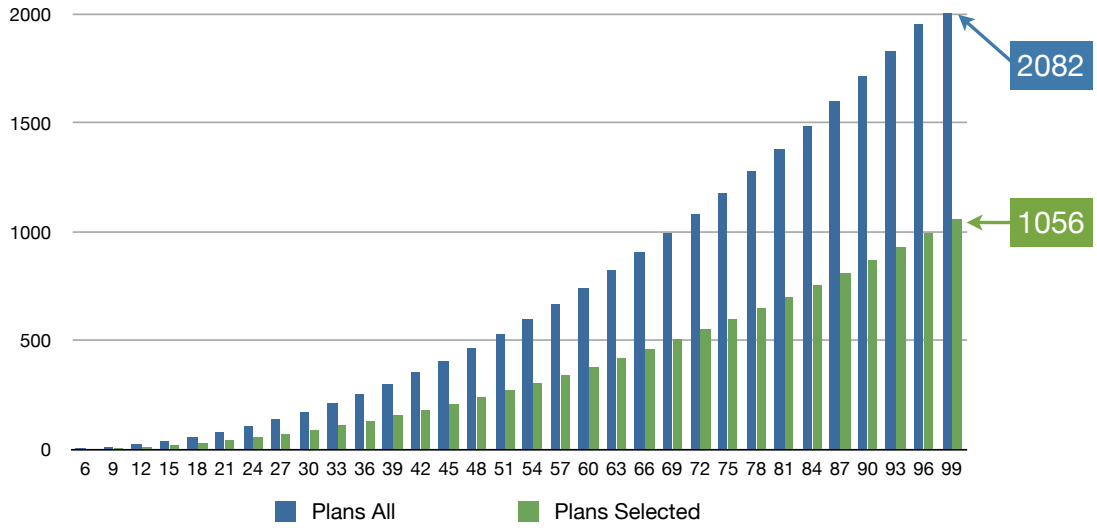

図 5.3: 選択段階の前後の回避策候補数

最適回避策提示法の選択段階は，現実に回避不可能な回避策候補や，サービスレベル違反が想定される回避策候補を提示しないための仕組みであった．図より，選択段階が正しく動作し，万が一不適切な回避策が推論結果に出現した場合にも対処し得るシステムである事が分かる．

選択段階の次の段階として，評価値を算出して最適な回避策を決定する評価段階がある．最終的に評価値を付与して出力された例を表 5.6 に示す．この例は，規模 10 (構成要素数 33) に基本ルール 1 〜 4 及びテンプレート 1 〜 9 を入力した際に得た例である．

表 5.6: 評価値算出結果

評価値	ルール	組合せ	操作
246.9	B1-詳細化	9	HOST-FS{2 or 3 or...} へ切り替える
243.4	B4-詳細化	100	HOST-Web{1 or 2 or...} のコア割当てを減らす
195.9	T2-詳細化	1	HOST-FS1 を再起動する

どれもサービスレベルを維持可能で，予測時間以内に回避できる回避策であるが，よりサービスレベルに影響が少ないものが高い評価値を得ている事が読み取れる．ただし，期待通りのランキングを得るためには，事前に正しいコスト情報を入力した上で，適切な基準値と重み付け係数を設定する必要がある．もっとも，コスト情報は熟練でなくとも収集できる上，基準値は運用ポリシーに含まれる情報であり，重み付け係数はこれらから計算で導出できる．

5.4　性能評価

性能評価に関しては，まずアルゴリズムの特徴や性質を詳細に分析するために項 5.4.1 にて処理ステップ毎の実行時間の傾向を調査し，次に項 5.4.2 にて入力したルール毎の実行時間を調査，最後に項 5.4.3 にて各ルールと処理ステップの関係について調査する．

5.4.1　各処理ステップの実行時間の比較

まず 200 台規模を想定した際の大まかな傾向を把握するため，複数のルール (基本ルール 1，基本ルール 2，及びテンプレート 1〜9) を同時に入力した際に算出した実行時間を図 5.4 にまとめた．横軸は構成要素数，縦軸は実行時間 (秒) を表す．

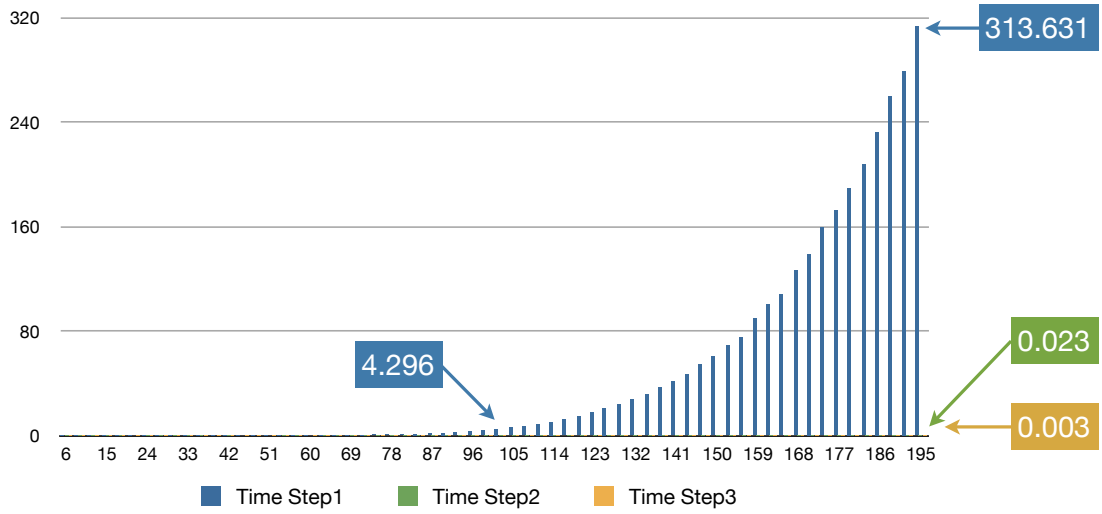

図 5.4: 各処理ステップの実行時間の比較

構成要素数が増えるにつれ，STEP1 (ルール詳細化) の実行時間が大きく上昇する傾向が把握できる．ルール詳細化アルゴリズムは全ての組合わせを探索する都合上，指数時間の計算量になっており，評価実験でもその性質が現れたと分かる．

一方で，STEP2 (回避策推論) 及び STEP3 (最適回避策提示法) は規模が増大した場合でも極めて短時間で実行を終えていることが分かる．

5.4.2　入力したルール毎の実行時間の比較

ルール毎に要した実行時間を比較するため，掛かった時間を片対数グラフで示したものを図 5.5 に示す．横軸は構成要素数，縦軸は実行時間 (秒) を表す．最大規模は約 100 台．

テンプレート集ではほとんど計算時間の増加傾向が掴めないが，その他の 4 つの基本ルールに関しては，いずれも程度は大きく異なるものの指数的な計算時間の増加を見せて

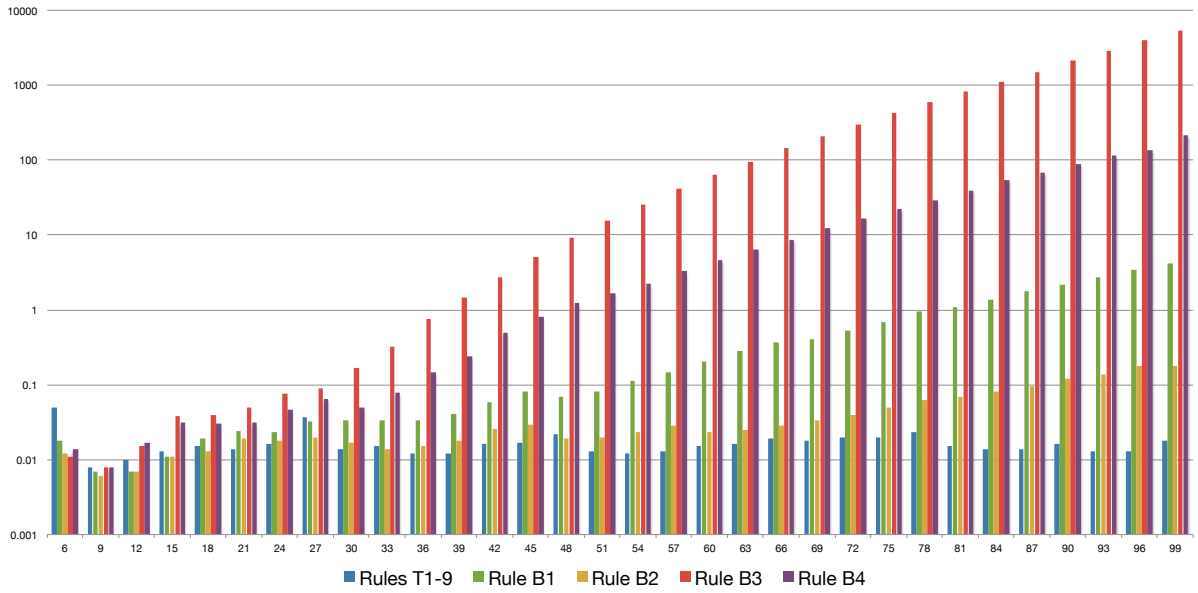

図 5.5: 各ルールの実行時間の比較

いる事がわかる．なお，0.01 前後の値には細かい増減が見受けられるが，通常のパーソナルコンピューターで実行している都合上，この程度の増減は誤差と考えられる．

5.4.3 ルールと処理ステップの関係

更に詳しく分析するために，ルールの特徴に応じた各処理ステップの実行時間の増加傾向を調査した．ルール毎の各処理ステップの実行時間の比較を片対数グラフにまとめ，並べたものを図 5.6 に示す．横軸は構成要素数，縦軸は実行時間 (秒) を表す．なお，項 5.4.2 にてテンプレート集の実行時間の傾向が想定した規模では把握できないほど小さかったことを踏まえ，テンプレート集に限り 1,000 台規模まで想定を拡大した再実験を行い，その結果をグラフにした．なお，各グラフの対数目盛りは揃えてある．

元々処理時間の大半が STEP1 で占められているため，項 5.4.2 同等の傾向が STEP1 の傾向にそのまま現れている．しかし複雑性の高いルールである基本ルール 3, 4 に限っては STEP2 の増加傾向も確認できる．規模を拡大したテンプレート集に関しては，ゆるやかな増加傾向が STEP1, STEP2 において初めて確認できた．STEP3 に関しては，全てのルールで目立った傾向は確認できなかった．

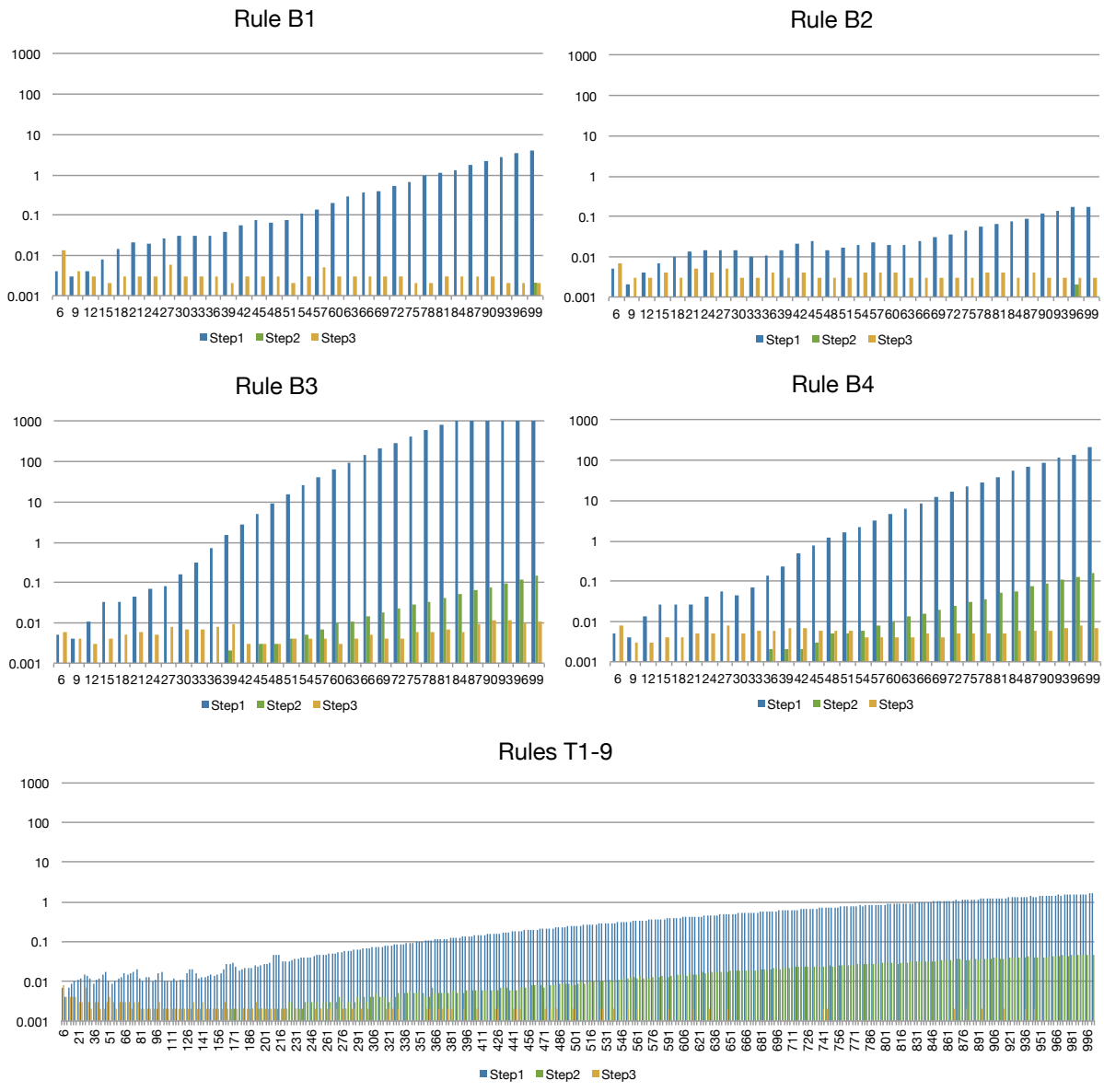

図 5.6: ルール毎の各処理ステップの実行時間の比較

第6章　考察

本章では，評価実験をふまえ提案手法の利点について大規模化への対応，複雑化への対応，そして熟練者への依存脱却の 3 つの観点から考察する．

6.1　大規模化への対応

システムが大規模化した際の懸念として，ルール数やルールの関係が膨大になり管理しきれなくなり，運用コストの増大や可用性の低下が懸念される事を指摘した．評価実験では，図 5.2 で示したように，構成要素数が 6 台の場合はルール入力が 13 件に対し詳細ルールは 24 件しか生成されておらず，削減量の割合としては 45.8% 程であるが，構成要素数を 99 台まで増加させると，削減量の割合は 99.9% を超え，圧倒的な削減量となる．この事から，同種の構成要素が多数配置されるような情報システムで，今回の入力のような抽象度の高いルールを記述することで，規模に応じたコスト増大を大きく抑制できるといえる．

さらに，あらゆる組合わせを網羅した詳細ルールを容易に生成できることは，より安全性の高い運用を迅速に展開できる事を意味する．従来は手作業で運用管理者の記憶の範囲でしかカバレッジされなかったルールメンテナンスを，予め全ての組合わせを導出できる本手法に切り替えることで，想定外の組合わせによる回避機会の喪失や，記入漏れによる運用失敗を防止するといった副次効果が得られる．

評価実験では，実行時間の調査においても大規模化に適応し得ると考えられる結果が得られた．図 5.6 から想定構成要素数 99 台の場面の各処理ステップの実行時間を抽出してまとめたものを表 6.1 に示す (太字は各処理ステップの最大値) .

STEP1 (ルール詳細化) に注目すると，基本ルール 3 と 4 の実行時間が突出しているが，それ以外のルールでは非常に高速に各処理ステップを終えている事が分かる．また，テンプレート 1〜9 は同時に 9 つのルールを入力した値であるが，図 5.6 を見ても明かなように計算量は線形時間である．つまり，基本ルール 3 や 4 といった複雑度の高いルールを避けて使用すれば，大規模な情報システムにも適用できると言える．

更にこれを裏付けるため，基本ルール 3 と 4 を除いたルールの集合と，基本ルール 3 と 4 を含めたルールの集合で実施した実行時間の比較を表 6.2 にまとめた．

基本ルール 3 と 4 で記述したような複雑性の高いルールが混入しないよう留意すれば，大規模環境にも十分適用しうる結果といえる．複雑性に関する詳しい議論は次節 6.2 で行うが，基本ルール 3 の中身は一つのルールに 3 種類以上もの構成要素について記述した

表 6.1: 各ルールのステップ毎の実行時間の比較

ルール名	構成要素種類数	切換	STEP1	STEP2	STEP3
テンプレート 1〜9	1	-	0.015	0.001	0.002
基本ルール 1	1	有	4.144	0.002	0.002
基本ルール 2	2	-	0.174	0.001	0.003
基本ルール 3	2	有	**5415.386**	0.143	0.001
基本ルール 4	3	-	213.302	**0.154**	**0.007**

表 6.2: 基本ルール 3 と 4 を含めた場合と含めない場合のステップ毎の実行時間の比較

ルール集合	構成要素種類数	切換	STEP1	STEP2	STEP3
T1-9 + B1-2	1-2	有	4.272	0.003	0.002
T1-9 + B1-4	1-3	有	5529.171	0.672	0.119

条件を含み，なおかつ切換表現も記述されたルールである．本評価実験では，図 5.6 で示したように，どのようなルールがどれほどの計算量になるかが明らかになったため，ルールの記入段階で計算量の予測を示し，危険なものは条件を分割して一つのルールで記述する構成要素種類数を抑えるよう修正を促すことで，組合わせ爆発を予防しながら大規模化に対応する事ができるのである．

6.2 複雑化への対応

今日の情報システムは構成や採用技術が複雑化し，また多様なニーズに応えるため運用ポリシーの複雑化も問題となっている点を指摘した．評価実験では，表 5.3 で説明した通り様々なパターンの運用ルールを想定し，同時に入力するルール数も変えながら評価を行った．しかし，実装したルール詳細化アルゴリズムの性質により，構成要素数が大規模で，かつ複雑なルールを展開した際，ルール詳細化に膨大な計算時間が必要になるという傾向も明らかになった．

前節 6.1 に示した各ルールのステップ毎の実行時間の比較表 6.1 に再び注目すると，基本ルール 3 の実行時間が著しいほか，基本ルール 4 においても基本ルール 3 程ではないが長い実行時間を要している．これらの事実から，各ルールの組合わせ爆発の起こしやすさは，

テンプレート 1〜9 < 基本ルール 2 < 基本ルール 1 < 基本ルール 4 < 基本ルール 3

という事がわかる．これを表 5.3 を参考にして一つのルールあたりの構成要素種類数を数字，切換の有無を Y/N (Yes/No) で置き換えると，

$$1N < 2N < 1Y < 3N < 2Y$$

となる．つまり，もっとも複雑性に影響する要因は切換，次に影響する要因が一つのルールあたりの構成要素種類数という事になる．実際の運用を想定した場合，切換という機能は不可欠であろう．

しかし，一つのルールあたりの構成要素種類数については，提案システムで用いたエキスパートシステムの特性を活用してルールを分割することにより，この値を 1 まで落とすことができる．エキスパートシステムの推論エンジンは，連鎖的な事実群生成と条件適用を新規事実が発見できなくなるまで繰り返す仕組みであるからである．

例えば，図 6.1 のテストルール 1 のようなルールがあったとする．前述の複雑度の表現に当てはめると，このルールは (抽象記述の) 構成要素種類数が 3，さらに切換表現が含まれるため $3Y$ となる．つまり (条件記述数は少ないが) 複雑性は高い．

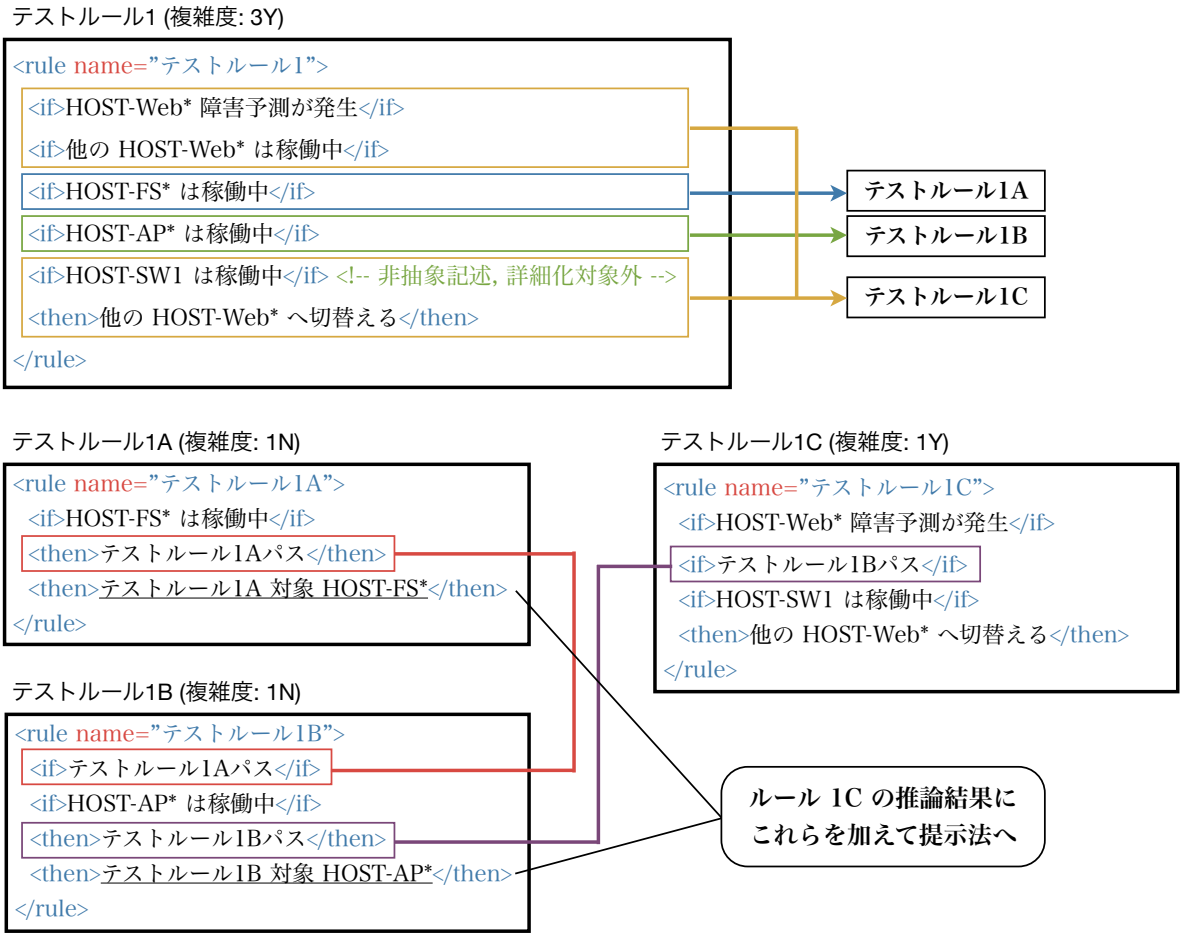

図 6.1: ルール分割を用いた高複雑度の解消

しかし，これを同図下部で示すように 3 つのルールに分割すると，複雑度は $\{1N, 1N, 2Y\}$ なる．複雑度が低いルールが多数ある環境で実験した結果は，複雑度が高いルールを少数ある環境よりも圧倒的に高速であることは，図 5.6 でテンプレート集 (複雑度 $1N$ を同時に 9 つ入力) のグラフとそれ以外 (複雑度 $1Y$ 以上を 1 つ入力) を比較すれば明らかである．このルール分割を行うことで，全ての基本ルールが $1N$ ないしは $1Y$ となり，複雑性の高いルールによる組合わせ爆発問題は大幅に改善する．

分割に必要なルール数は，記述種類数と同数となり，ルール記述の冗長性が増加することになる．この点に関しては，分割・統合は自動化することによってルール記述表現力を維持しながら複雑度を抑える事で解決できると考えられる上，分割や統合に必要な計算量は線形時間になると見込まれる．

さて，依然として残る問題が複雑度 $1Y$ は指数時間の計算量という点である．例えば，表 6.1 に示したように，基本ルール 1 の実行時間が構成要素数約 100 台の環境での全処理ステップの合計で 4 秒余りであったが，これを 200 台規模の環境で再実験したところ 313.657 秒を要した．内訳のほぼ全てが STEP1 (ルール詳細化) であり，膨大な実行時間への懸念はぬぐえない．そこで，この時間を更に圧縮することは可能かを議論したい．

ここでは，これまで説明してきた提案手法および評価システムが以下の 3 つの特徴を有す構造である事に注目したい．

1. 評価システムが STEP1, STEP2, STEP3 の 3 ステップで構成されている．

2. 構成が変わらなければ，STEP1 の出力は再利用できる．

3. 入力は，小さな情報 (既知事実またはルール) の集合である．

特徴 1 と 2 をふまえると，システムへの初期導入時のみ STEP1 〜 3 を実行し，STEP1 の出力 (詳細ルール一覧) を XML 形式で保存しておくことで，以後構成が変わるまでは障害予測が出るたびに STEP1 の出力を読み込み，STEP2 から処理をスタートできる．STEP2 や STEP3 はほぼ線形時間で処理が可能なため，大規模化，複雑化の双方に適応しうる運用が実現できるといえる．

さらに，特徴 3 を言い換えると，部分入力，部分出力が可能という事になる．構成が変化した際に STEP1 を最初から全てやり直すのではなく，部分的な修正を行うことで，STEP1 の再実行時間を大幅に圧縮できる．ただし，再生成に必要な時間は，構成要素の数から単純に見積もることはできない．例えば，構成要素が 1,000 台の環境が存在し，基本ルールが 50 件存在する場面で，そのうち 100 台の構成が何らかの要因で変化したとしよう．この場合必要な計算量は元々の 1/10 とはならず，構成変化部分に言及がある基本ルールの計算量の合計となる．構成変化に含まれる要素に 20 の基本ルールが言及していれば，たとえ構成変化が全体の 10% だったとしても，書き換え対象は基本ルール数ベースで 40% になり，その分の詳細化を実施して古い構成要素に関する記述を置き換える必要がある．しかし，全てを 1 からやり直すよりも多くの時間を圧縮できる事は明らかであり，有用な時間圧縮手段となるだろう．

6.3 熟練者への依存脱却

従来，障害の解決や回避には幅広い知識や豊富な経験を持った運用管理者が活躍してきた．しかし，そうした大規模・複雑な情報システムが多数必要とされる次代になり，運用管理者の負担の増加や依存は増している．本手法では，そうした熟練者への依存脱却を大きな目標として提案するものであり，評価実験においても，構成把握や高い知識を必要とせずに適した回避策が生成されるという一連の動作を示した．評価システムの入力データは，運用ポリシーに関する情報以外はほとんどが自動的に収集可能であり，更に自動化を押し進める事が可能である．

また提案手法は，スキルや経験の浅い人にありがちな様々なミスにも対処できるように考慮した．表 6.3 にミスの種類とその対処についてまとめた．

表 6.3: 想定する入力ミスとその対処

ミスの種類	対処される場所	備考
ルール記述の構文ミス	ルール読込部	XML 文書型定義で検証
ルールの論理的不整合	推論エンジン	どの推論にも適用されない為
存在する誤った構成要素記述	選択段階	規定外の影響がある場合
存在しない誤った構成要素記述	ルール詳細化	構成要素と合わず展開不可

ただし，ルール記述のうち論理判定に影響しない部分の記述 (回避操作手順の記述ミスなど) には，提案手法では回避できない．これには，提案システムよりも前の段階で防ぐことが必要になるだろう．しかし，ルール記述マニュアルを用意したり，入力インタフェースを工夫したりして対処可能であると考えられる．

このほか，過度な自動化，特にブラックボックス化は運用管理者の成長を阻害するという弊害も指摘される．そこで提案手法の検討や評価システムの実装においては，入力したデータから提示結果に至るまでの処理過程を可能な限り視覚的に把握できるよう留意した．特に，回避策推論システムにおいてエキスパートシステムを採用した理由の一つに，自然言語で書かれた事実群やルールが連鎖的に適用され推論される様子が見えるため動作が分かりやすいという特徴が挙げられる．これは，障害の分析を助けるだけでなく，運用管理者が動作を理解しながら利用できるという利点がある．

最適回避策の提示においては，サービスレベルやコストといった判断指標を考慮し，さらに評価値を算出・用いることで，経験の浅い運用管理者でも効果の見込める回避策を素早く把握，実行することが可能になった．また，なぜその回避策が最適なのかが把握できるため，熟練者に近い判断を提案システムから学ぶ事にも繋がり，熟練者に依存しない教育の支援にも繋がると期待できる．

第7章　おわりに

本章では，これまでのまとめを述べた後，今後の課題について述べる．

7.1　まとめ

本研究では，大規模・複雑化の一途を辿る情報システムにおいて，熟練の運用管理者に依存せずに障害を回避するという高度な運用措置の支援を提供する仕組みを提案した．節 3.1 では自動推論システムを用いて運用ルールから障害回避策を得る予備実験を通して，ルール数爆発，判断指標考慮，回避策安全性の 3 つを解決すべき問題点として挙げ，ルール数爆発の問題に対する解決策として節 4.2 でルール生成自動化を，判断指標考慮及び回避策安全性の問題に対する解決策として節 4.3 で最適回避策提示法をそれぞれ提案した．

ルール生成自動化では，運用管理者の負担を最小限にするための工夫としてルール詳細化アルゴリズムを考案し，構成要素数に依存しない抽象度の高いルール記述を可能にした．これにより，ルールを記入・入力する手間を減らすだけでなく，構成や状態を正確に把握するスキルがない運用管理者でもルールのメンテナンスに加わる事ができるといった利点が挙げられる．なおかつ第 6 章では，全ての組合わせを網羅した詳細ルールを容易に展開できる事により，想定外の組合わせによる回避機会の喪失や，記入漏れによる運用失敗を防止できるという利点にも言及した．第 5 では，実際に十数件のルールから規模に応じて多数の詳細ルールが展開され，大規模な構成では 99% 以上が詳細ルールで占められる様子を紹介した．

しかし，複雑なルールを入力した際にルール詳細化アルゴリズムの実行時間が膨大になる問題が明らかになり，特に規模の大きさと複雑さが重なった場合には現実的な時間に収まらないほどの計算量となった．そこで第 6 章にてこの問題の回避法について議論し，推論システムの特徴を活かして複雑度の高いルールを複数に分割し複雑性を下げるアイデアや，ルール詳細化アルゴリズムの実行結果を次回以降の実行用に再利用するアイデア，更に構成変化時の部分的な再構成というアイデアを提示した．これらにより，複雑度に応じて組合わせ爆発の程度まで大きくなる問題を回避したほか，極めて大規模な環境を想定した場合でも，現実的な計算時間に収められる見通しを示した．

最適回避策提示法では，猶予時間などの現在の状況に応じて最適回避策を決定，提示するための仕組みとして最適回避策提示アルゴリズムを考案し，知識や経験の浅い運用管理者でも常に安全で最適な障害回避策を得られる運用支援を可能にした．アルゴリズムは選択段階，評価段階，提示段階の 3 段階で構成されるが，最後の提示段階では評価段階で

算出した評価値でランキングした結果を運用管理者に提示し，なぜその回避策が最適なのかを代替案と比較しながら把握できる仕組みとした．これにより，知識や経験の浅い運用管理者でも迷わず最適回避策の実行の着手が可能になる．第 5 章では，ルール毎にコスト情報を集めて実現性判断，評価値の算出を行い，運用ポリシーに基づいた最適回避策と代替案のランキングが提示される様子を紹介した．性能評価においては，計算量は線形時間であり，所要時間も想定した規模の範囲ではあらゆる状況下で 10 ミリ秒を下回る軽微な水準であった．

これらの結果や議論を総括すると，提案手法は障害予測における最適な障害回避手段を提示できる有用な手法であるといえる．提案手法を用いると，経験やスキルが浅い運用管理者でも障害予測からの障害回避が容易に実現できるようになる．障害回避を 1 件成功させることで，その障害の発生によって引き起こされる可能性がある第 2，第 3 の障害を回避することにも繋がるため，障害回避が情報システムの可用性向上にもたらす効果は極めて大きい．また，熟練管理者に恵まれていない，育っていない会社などの組織や，発展途上国などの地域は世の中に数多く存在し，プライベートクラウドコンピューティングなどが更に世界中で用いられるようになると，今以上に大きな問題になる事が考えられる．そうした状況下を第一に想定した本手法は，近い将来を見据えた上で不可欠な手法になる事も考えられるであろう．

7.2 今後の課題

今後の課題としては，まず提案手法を実装した評価システムにおける各アルゴリズムの性能向上が挙げられる．障害予測から素早く回避操作を導出できればできるほど，運用管理者は余裕を持って回避操作の実行にあたることができる．そのためには，線形時間以下の計算量を目指したアルゴリズム性能の改良が望ましい．

加えて，実際の障害予測技術を用いた実験の検証が挙げられる．本研究では，障害予測情報が正確であればあるほど，効果的な回避策を提示できると考えられる．しかし現状では，あらゆる情報システムに適用可能な汎用的で高精度な障害予測技術は確立されていない．そのため，既存の障害予測技術の中からなるべく精度の高いものを選択，組み合わせる必要がある．Gainaru ら [12] による統計的・データマイニング的手法を HPC システムに適用した例などを参考にしたい．

謝辞

本研究を進めるにあたり，情報社会基盤研究センター 敷田幹文教授には日頃よりご指導，ご助言をいただき心より深く感謝致します．また本研究に多くの技術的・専門的助言を頂いた，敷田研究室の窪田清氏，坂下幸徳氏，情報処理学会インターネットと運用技術研究会の皆様，及び情報社会基盤研究センター技術職員の皆様に感謝致します．最後に，自主勉強会の開催を通じて活発な技術交流や議論にお付き合い頂いた主催者の並河雄紀君と参加者の皆様，そして多くの励ましを頂いた同期，先輩，後輩，家族に感謝致します．

研究業績

口頭発表および論文集掲載 (査読あり)

加藤裕，敷田幹文：障害予測における最適な障害回避手段の提示法，第 5 回 情報処理学会インターネットと運用技術シンポジウム (IOTS2012) 論文集，pp. 110-116 (2012)

参考文献

[1] International Business Machines Corporation. Tivoli. http://www-01.ibm.com/software/tivoli/.

[2] Ramendra K. Sahoo, Adam J. Oliner, Irina Rish, Manish Gupta, José E. Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In Lise Getoor, Ted E. Senator, Pedro Domingos, and Christos Faloutsos, editors, *KDD*, pp. 426–435. ACM, 2003.

[3] Peter Bodik, Greg Friedman, Lukas Biewald, Helen Levine, George Candea, Kayur Patel, Gilman Tolle, Jonathan Hui, Armando Fox, Michael I. Jordan, David A. Patterson, and David A. Patterson. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *ICAC*, pp. 89–100, 2005.

[4] 日本ヒューレット・パッカード株式会社. Hp service intelligence. http://www8.hp.com/jp/ja/hp-news/article_detail.html?compURI=tcm:191-1198948.

[5] International Business Machines Corporation. Puresystems. http://www.ibm.com/ibm/puresystems/.

[6] Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat. *Building expert systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[7] Agnar Aamodt and Enric Plaza. Case-based reasoning; foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS*, Vol. 7, No. 1, pp. 39–59, 1994.

[8] 本村陽一, 岩崎弘利. ベイジアンネットワーク技術 ユーザ・顧客のモデル化と不確実性推論. 東京電機大学出版局, 2006.

[9] Oracle Corporation. Java se overview - oracle technology network. http://www.oracle.com/technetwork/java/javase/overview/index.html.

[10] 中村友洋. Web アプリケーションの障害を予測する “アクセス時間解析方式” の提案. 情報処理学会論文誌. コンピューティングシステム, Vol. 47, No. 12, pp. 349–357, Sep 2006.

[11] Dean Wampler and Alex Payne. *Programming Scala*. O'Reilly Media, 2009.

[12] Ana Gainaru, Franck Cappello, Joshi Fullop, Stefan Trausan-Matu, and William Kramer. Adaptive event prediction strategy with dynamic time window for large-scale hpc systems. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, SLAML '11, pp. 4:1–4:8, New York, NY, USA, 2011. ACM.

付 録A　ルール詳細化アルゴリズム記述

第 4 章の節 4.2 で述べたルール詳細化アルゴリズムの大まかな記述をコードリスト A.1 に示す．なお，この記述は計算量を検証・理解するために用意したもので，プログラムの主要な記述のみを取りあげたダイジェストである事に注意されたい．

コードリスト A.1: ルール詳細化アルゴリズム記述

```
1  // Classes
2  Rule rule;  // This class provides data & functions of the rule
3  Host host;  // This class provides data & functions of the host
4  HostPair pair;  // Paring of host and alternate host
5
6  // Variables
7  List<Rules> *Rules;  // Intermediate data
8  Set<Host> hosts, subHosts;  // Collection of hosts
9  Map<String, List<HostPair>> alternations; // Expanding patterns
10 List<String> ifList, thenList;  // if/then elements of the rule
11 prefix: String;  // Abstract description
12
13 // Constants
14 final String TGT; // Tag for recognizing hosts
15 final String ALT; // Tag for recognizing alternate hosts
16
17 for (Rule abstractRule : rules) {
18   ifList = abstractRule.getIfList();
19   thenList = abstractRule.getThenList();
20   abstractRule = tagging(abstractRule);
21   alternations = createAlternations();  // Create list of hosts
22   for (Host target : hosts) {
23     prefix = getPrefix(alternations, target);
24     subHosts = deepCopy(hosts).remove(target);
25     for (Host subTarget : subHosts) {
26       if (subTarget.getName().startsWith(prefix)) continue;
27       alternations.get(prefix).add(new HostPair(target,
```

```
        subTarget));
28    }
29  }
30  expandedRules.add(abstractRule);  // Set initial rule
31  for (Map.Entry<String, List<HostPair>> entry : alternations.
      entries()) {
32    midRules.clear();
33    midRules.addAll(expandedRules);
34    for (HostPair pair : entry.getValue()) {
35      tempMidRules = deepCopy(midRules);
36      for (Rule rule : tempMidRules) {
37        Rule expandedRule = new Rule(...);
38        for (String ifText : rule.getIfList()) {
39          if (ifText.contains(TGT)) {
40            prefix = getHostName(ifText);
41            if (pair.get1st().getName().startsWith(prefix)) {
42              expandedRule.addIf(
43                  unTagging(prefix, pair.get1st()));
44            } else {
45              expandedRule.addIf(ifText);
46            }
47          } else if (ifText.contains(ALT)) {
48            prefix = getHostName(ifText);
49            if (pair.get2nd().getName().startsWith(prefix)) {
50              expandedRule.addIf(
51                  unTagging(prefix, pair.get1st()));
52            } else {
53              expandedRule.addIf(ifText);
54            }
55          } else {
56            expandedRule.addIf(ifText);
57          }
58        }
59        for (String thenText : rule.getThenList()) {
60          if (thenText.contains(TGT)) {
61            prefix = getHostName(thenText);
62            if (pair.get1st().getName().startsWith(prefix)) {
63              expandedRule.addThen(
```

```
64                      unTagging(prefix, pair.get2nd()));
65                  } else {
66                    expandedRule.addThen(thenText);
67                  }
68                } else if (thenText.contains(ALT)) {
69                  prefix = getHostName(thenText);
70                  if (pair.get2nd().getName().startsWith(prefix)) {
71                    expandedRule.addThen(
72                        unTagging(prefix, pair.get2nd()));
73                  } else {
74                    expandedRule.addThen(thenText);
75                  }
76                } else {
77                  expandedRule.addThen(thenText);
78                }
79            }
80          }
81        }
82        for (Rule tempRule : tempMidRules) {
83          if (!midRules.contains(tempRule))
84            midRules.add(tempRule); // Add intermediate data
85        }
86        midRules.removeAll(expandedRules);
87        expandedRules = midRules;
88      }
89      rules.addAll(expandedRules);
90    }
91    return rules;
```

付 録B　最適策提示アルゴリズム記述

第 4 章の節 4.3 で述べた最適回避策提示アルゴリズムの大まかな記述をコードリスト B.1 に示す．なお，この記述は計算量を検証・理解するために用意したもので，プログラムの主要な記述のみを取りあげたダイジェストである事に注意されたい．

コードリスト B.1: 最適策提示アルゴリズム記述

```
1  // Classes
2  Plan plan; // Abstraction of workaround plan
3  Cost cost; // Time, Monetary cost, Performance and reliability
4  Weight weight; // Weight coefficient for costs
5  Prediction prediction; // Collection of prediction facts
6
7  // Variables
8  List<String> resultFacts; // Result from Inference Engine
9
10 // Constants
11 Cost BASIS; // Basis data from configuration file
12 Weight WEIGHT;  // Weight data from configuration file
13
14 // Create abstract classes from the Engine result
15 for (String fact : resultFacts) {
16   if (Prediction.isPrediction(fact)) {
17     prediction.set(fact);
18   } else if (Prediction.isReason(fact)) {
19     prediction.setReason(fact);
20   } else if (isInferenceResult(fact)){
21     for (Plan tempPlan : plans) {
22       if (tempPlan.getRuleName().equals(getRuleName(fact))) {
23         plan = tempPlan;
24         break;
25       }
26     }
27     plan.addAction(getAction(fact));
```

```
28      plans.add(plan);
29    }
30  }
31  // Add cost to plan
32  for (Plan plan : plans) {
33    for (String action : plan.getActions()) {
34      for (Cost cost : costs) {
35        if (cost.isMatch(action)) {
36          plan.addcost(cost);
37          break;
38        }
39      }
40    }
41  }
42  // Selecting step
43  // (T: Time, M: MonetaryCosts, P: Performance, R: Reliability)
44  for (Plan plan : plans) {
45    if (plan.getTotalT() > basis.getT() ||
46        plan.getTotalM() > basis.getM() ||
47        plan.getTotalP() < basis.getP() ||
48        plan.getTotalR() < basis.getR()) {
49      continue;
50    }
51    selectedPlans.add(plan);
52  }
53  // Scoring step
54  for (Plan plan : selectedPlans) {
55    plan.setScore(
56        weight.getTWeight() * (getTotalT() - basis.getT()) +
57        weight.getMWeight() * (getTotalM() - basis.getM()) +
58        weight.getPWeight() * (getTotalP() - basis.getP()) +
59        weight.getRWeight() * (getTotalR() - basis.getR()));
60  }
61  // Presentation step (abbr.)
62  return selectedPlans;
```