情報処理学会論文誌 Vol.56 No.5 1363–1376 (May 2015)

EVMを利用したDBCアプリケーションのための適応フレームワーク

末永 俊一郎[1,a)] 鄭 顕志[1,b)]

受付日 2014年8月18日, 採録日 2015年2月4日

概要：期限，総予算の制約のもと，タスクを実行する *DBC*（Deadline Budget Constrained）アプリケーションは，開発時に想定した環境と実行時の環境が同等であれば，期限と総予算の制約を守ることができる．一方で，予期できない変化が生じると，制約を守れなくなる場合があるため，計算機器を追加する等の変更を行って変化に適応する必要がある．適応時には，*DBC* アプリケーションの実行状況の分析が重要となる．既存手法を拡張した分析手法では，タスク実行に要する予算の重みを考慮しないことから，異なる変更を行いたい 2 つの状況を同一の状況と分析してしまうという課題がある．この課題に対して，プロジェクトマネジメントの進捗管理，予算管理で用いられる EVM（Earned Value Management）による分析手法を提案する．また，*DBC* アプリケーションの状況を EVM に基づき分析し，その分析結果に基づき *DBC* アプリケーションの設定や実行環境を変更する適応フレームワークを提案する．提案フレームワークを実装し，シミュレーション環境で評価することにより，提案手法が既存手法に比較し，*DBC* アプリケーションのために適した分析手法であること，提案フレームワークの適応範囲を示す．

キーワード：進捗，予算，アーンドバリューマネジメント，適応

Earned Value Management Based Adaptation Framework for Deadline Budget Constrained Applications

Shunichiro Suenaga[1,a)] Kenji Tei[1,b)]

Received: August 18, 2014, Accepted: February 4, 2015

Abstract: *DBC* (Deadline and Budget Constrained) applications which have deadline and budget constraints can keep these constraints if the execution environments are same as that of assumed environments in development time. However, unexpected variations introduce possibilities that *DBC* applications violate these constraints. Therefore, *DBC* applications perform modifications to adapt to situations, for example, adding computational resources. Methods to analyze the statuses of *DBC* application is one of the key success factors in adaptation. A representative method of existing works can not distinguish statuses which needs different modifications, because the method do not consider the weight of budget to execute a task. To overcome this problem, we propose an EVM (Earned Value Management) based analysis method. EVM is a method to manage budget and progress in project management. In addition, we propose an adaptation framework which analyzes the statuses of *DBC* applications and perform modifications to applications and execution environments. We implemented the proposed framework and evaluated the framework in the simulation environment. We show that the proposed method is more suitable analysis method than that of existing methods. We also show adaptation coverage of the proposed framework.

Keywords: progress, budget, earned value management, adaptation

[1] 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] suenaga@nii.ac.jp
[b)] tei@nii.ac.jp

1. はじめに

1.1 DBCアプリケーションとその開発

インターネット上で提供されるサービスを実現するア

プリケーションの中で，期限内，総予算内でサービスを提供するアプリケーションは，*DBC*（Deadline and Budget Constrained）アプリケーションと呼ばれている [7]．*DBC* アプリケーションの例に，IaaS を用いたデータ解析アプリケーションや，WSN（Wireless Sensor Network）を用いたモニタリングアプリケーションがある．

DBC アプリケーションの開発者は，実行環境や，利用する計算機器のパフォーマンスを想定し，期限内，総予算内でタスクを実行する *DBC* アプリケーションを開発しなければならない．期限内，総予算内でタスクが完了するように，タスクをスケジューリングする手法 [18], [23] や計算機器を動的に割り当てる手法 [1], [3] 等が提案されている．このような手法を用いて開発された *DBC* アプリケーションは，実行環境と計算機器のパフォーマンスが開発時の想定と同じであれば，実行時にも期限と総予算の制約を満たすことができる．

1.2 DBC アプリケーションの実行と適応フレームワーク

DBC アプリケーションの実行中に，開発時には詳細まで把握しきれない変化が生じた場合には，制約を守れない場合がある．たとえば，クラウドコンピューティングでは，VM のパフォーマンスや，VM の利用開始までのレイテンシが一定でないことが知られている [13]．また，WSN では，バッテリ切れによるノードの離脱やパケットロスがあることが知られている [12]．こうした予期できない変化が生じ，積み重なることで，期限内に実行すべきタスクを完了できない場合がある．また，期限内にタスクを完了させるために計算機器を追加すると，総予算を超過する場合もある．

DBC アプリケーションの制約を守るためには，期限と総予算に対する実行中の状況を分析し，必要に応じて変更を加える必要がある．たとえば，データ解析アプリケーションにおいて，VM のパフォーマンスが開発時の想定より低い場合には，期限を守れない状況となりうる．このような状況では，新たな VM を追加し実行時間を短縮する等の変更により改善を試みる必要がある．ただし，こうした VM の追加は，実行時間を短縮する一方で，予算消化を増加させる．たとえば，VM を追加した結果，期限は守れるが総予算は守れない状況になりうる．このような状況では，利用している VM をより金額の安い VM に切り替え，消化する予算を削減する等の変更により改善を試みる必要がある．こうした変更をマニュアルで実施することは，アプリケーション開発者の負担となる．本研究では，変更を容易にするため，*DBC* アプリケーションの制約を守る適応フレームワークの構築を目的とする．

1.3 分析手法の重要性と既存研究の課題

誤った分析に基づく変更は，制約を満たせなくなる可能性を増加させるため，適応フレームワークにおける状況の分析方法は重要となる．期限と総予算の制約がある *DBC* アプリケーションの状況分析方法は，次の 2 つの要求を満たす必要がある．(a) 開発時および実行中に得られる情報から，期限の制約を守れるか分析できること．(b) 開発時および実行中に得られる情報から，総予算の制約を守れるか分析できること．

既存研究 [5], [30], [31] では，期限の制約を守るために，タスクの処理量の計画値（以降，「計画進捗」）と実績値（以降，「進捗」）の差分をしきい値と比較することで，期限の制約を守れるか分析する手法を提案している．既存手法を拡張し，計画進捗と進捗に加え，予算の計画値（以降，「計画予算」）と実績値（以降，「消化予算」）を比較分析する手法では，要求 (a) を満たしても要求 (b) を満たすことができない．既存手法の拡張では，異なる変更を行うべき 2 つの状況を，同一の状況と分析してしまうという課題がある．この結果，完了時に総予算を超過するといった弊害が生じる可能性がある．この課題は，計画進捗と進捗，計画予算と消化予算を個別に比較する際に，進捗を達成する際の予算の重み（タスク実行に要する予算の重み）を考慮していないことに由来する．

1.4 提案手法と本論文の構成

この課題を解決するため，我々は，プロジェクトマネジメントで利用されている進捗管理，予算管理の手法である EVM（Earned Value Management）を導入する．EVM では，計画進捗を金額で表すことで，計画進捗と計画予算を 1 つの指標に統合する．これに加えて，進捗も同様に金額で表すことで，進捗，計画予算（計画進捗），消化予算がすべて金額で表現される．これにより，進捗を達成する際の予算の重みが考慮される．EVM では，進捗と計画予算の比をとることで，期限の制約を守れるか分析できる．また，進捗と消化予算の比をとることで，総予算の制約を守れるか分析できる．これにより，異なる変更を行いたい状況を区別可能とする．

本研究では，*DBC* アプリケーションの状況を EVM に基づき分析し，その分析結果に従って *DBC* アプリケーションを変更する適応フレームワークを提案する．本フレームワークをモニタリングアプリケーションの例で実装し，シミュレーション環境において評価することで，総予算を超過する可能性が低下したこと，提案フレームワークの適応範囲を示す．

本論文の構成は次のとおりである．2 章で想定する事例および課題を述べる．3 章で EVM の着想の背景と EVM の説明を行う．4 章で提案手法，5 章で評価を記載する．6 章で議論を行い，7 章で関連研究との比較を行う．8 章で本論文をまとめる．

表 1 DBC アプリケーションの例
Table 1 Examples of DBC applications.

アプリケーション名称	概要	タスク	予算制約
モニタリングアプリケーション	WSN ノードを用いて計測を行い，データをネットワーク内部の特定のノードに保存する．	2 秒間隔で同時刻に取得された 3 つの異なるノードの温度データの平均値（データ A），最も近傍のノードの温度データ（データ B）を特定のノードに保存する．データ A とデータ B の合計データ数を 90%以上の正確さで保存する．	電力
データ解析アプリケーション	入力されたワークフローに基づき，データの解析処理を実行する．	ワークフローで定められたタスクを指定された実行順序に従って実行する．負荷に応じて複数台の VM で分散処理を行う．	金額

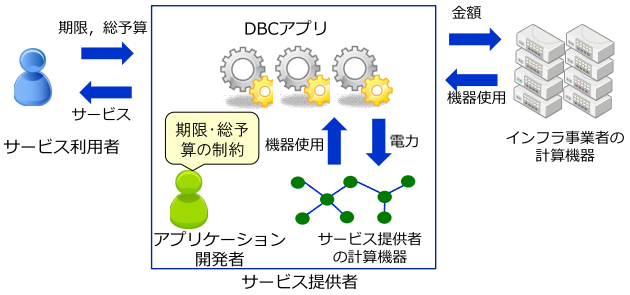

図 1 想定環境
Fig. 1 Assumed environment.

2. 想定する事例および課題

2.1 想定環境と DBC アプリケーション

本研究の想定環境を図 1 に示す．図 1 のとおり，本研究では，サービス利用者とサービス提供者を想定する．サービス提供者に存在するアプリケーション開発者は，サービスを実現する *DBC* アプリケーションを，IaaS 等のインフラ事業者の提供する計算機器や，WSN 等の自身の所有する計算機器を利用して開発する．このとき，計算機器の利用の対価として，前者の場合には金額，後者の場合には電力が発生する．アプリケーション開発者は，これらの計算機器の利用の対価を総予算内にとどめる必要がある．そこで，本研究では，予算の制約として金額と電力を考慮する．

表 1 に *DBC* アプリケーションの例を示す．モニタリングアプリケーションは，電力を予算制約とするタイプの *DBC* アプリケーションであり，期限内に温度データの計測と保存を行うアプリケーションである．また，データ解析アプリケーションは，金額を予算制約とするタイプの *DBC* アプリケーションであり，期限内にワークフローで与えられるデータの解析を行うアプリケーションである．本研究では，モニタリングアプリケーションを例にとり，説明を行う．

2.2 要求

DBC アプリケーションの実行時に，開発時には詳細まで把握できない変化が生じると，期限を守れなくなる場合がある．また，その状況を打開するために，計算機器を追加するといった変更を行うと総予算を守れなくなる場合もある．*DBC* アプリケーションの制約を守る適応フレームワークは，こうした状況を分析し，必要に応じて変更を行い変化に適応する必要がある．そこで，適応フレームワークは以下の要求を満たす必要がある．

要求 1) 状況の分析
DBC アプリケーションの実行状況を分析し，(a) 開発時および実行中に得られる情報から，期限の制約を守れるか分析できること．また，(b) 開発時および実行中に得られる情報から，総予算の制約を守れるか分析できること．

要求 2) アプリケーションへの変更
分析結果に基づきアプリケーションの振舞いを変更できること．また，計算機器を変更する等のアプリケーションの実行環境を変更できること．

2.3 課題

要求 1 を満たす手法として，大きく，計画値と実績値を比較して状況を分析する手法，制約とその予測値を比較して状況を分析する手法がある．前者は主にタスクスケジューリングの研究分野で行われてきた．たとえば，期限の制約を守る際に，計画進捗（タスク処理の計画時間）と進捗（タスク処理の実績時間）を比較し，その差分がしきい値を超えた場合には，期限が守れないと分析する手法 [29], [30], [31] が提案されている．後者は，主にサービスコンピューティングや自己適応の研究で実施されてきた．文献 [5], [15], [21] では，応答時間等の制約となる SLA の予測値を，実行中に取得された値と各種予測技術から算出し，予測値が SLA を超えると，SLA が守られないと分析する手法を提案している．双方の手法ともに，2 つの値を比較し，差分をみて分析する点は共通している．ただし，後者の手法は，予測するための統計データや，アプリケーションドメインに特化した予測方法を用いており，過去データが存在しない場合や，ドメインの異なるアプリケーション

表 2 既存手法の欠点の例：ケース 1 もケース 2 も同じ状況と分析される [1]．しかし，ケース 2 は総予算を超過する可能性がある [2]．ケース 1 とケース 2 は異なる状況である
Table 2 Examples of disadvantage of conventional approach.

ケース	計画進捗	進捗	計画予算	消化予算	分析結果	総予算	予想総予算（参考）
1	4 個	3 個	8 mJ	7 mJ	進捗が計画進捗を下回り，消化予算が計画予算を下回る [1]	16 mJ	16 mJ
2	4 個	3 個	8 mJ	7 mJ	進捗が計画進捗を下回り，消化予算が計画予算を下回る [1]	16 mJ	18 mJ[2]

に適用することは難しい．そこで，本研究では，前者の手法を参考にする．

DBC アプリケーションの場合，期限と総予算の 2 つの制約を守る必要がある．既存手法を拡張し，計画進捗と進捗に加えて，計画予算と予算を比較分析する手法は，要求 1 (a) を満たすが，要求 1 (b) を満たさず，次の課題をかかえる．

課題

進捗を達成する際の予算の重みを考慮せずに進捗と予算を個別に比較するため，重みを考慮した場合には異なる状況を，同一の状況と分析してしまう課題がある．この結果，完了時に総予算の制約が守られないという弊害が生じる可能性がある．

課題を説明するため，表 1 に示すモニタリングアプリケーションを簡易化した例を取り上げる．データ A を 3 つのノード，データ B を 1 つのノードで取得する．各データの取得には 1 mJ 必要とする．10 分以内，16 mJ 以内で，計 8 個（データ A を 4 個，データ B を 4 個）のデータを取得するとする．5 分経過時点での計画値は，8 mJ 消費し，計 4 個（データ A を 2 個，データ B を 2 個）のデータ取得となる．実行時には，7 mJ でデータ 3 個が取得されていた．**表 2** に示すとおり，進捗と計画進捗をデータ数で比較すると，進捗は 3 個，計画進捗は 4 個であり，進捗が計画進捗を下回る．また，7 mJ と 8 mJ で，消化予算が計画予算を下回る．この際，ケース 1 とケース 2 が存在しうる．単純に残りのデータ数を取得するための所要電力を加算すると，ケース 1 は総予算が 16 mJ と見積もられ，ケース 2 は 18 mJ と見積もられる．ケース 2 は総予算を超過する可能性が高く，ケース 1 とケース 2 は同じ状況ではない．この 2 つを，同じ状況と分析し，進捗を挽回する変更としてデータの再取得を行うと，ケース 2 の場合は総予算を超過する可能性がある．

3. EVM

3.1 EVM 適用の着想の背景

アプリケーション開発等の実際のプロジェクトでは，進捗管理と予算管理が行われている．プロジェクトマネジメントのデファクトスタンダードである PMBOK（Project Management of Body of Knowledge）[4] では，進捗管理と予算管理の手法として EVM を推奨している．EVM では，計画進捗を金額で表すことで，計画進捗と計画予算を統合する．また，進捗も金額で表すことにより，計画予算，進捗，消化予算をすべて同じ単位で表現する．EVM では，進捗，計画進捗を金額で表すことにより，進捗を達成する際の予算の重みを考慮するという特徴を持つ．計画予算，進捗，消化予算を比較することで，期限内，総予算内でのプロジェクトの完了可否を分析できる．具体的には，進捗と計画予算の比により，実行時点までに完了した作業の時間の効率で，期限の制約を守れるか分析できる．また，進捗と消化予算の比により，実行時点までに完了した作業の予算の効率で，総予算の制約を守れるか分析できる．我々は，EVM のこれらの性質が，*DBC* アプリケーションを適応させる際の分析手法として利用できるのではないかと考えた．

表 3 EVM の指標
Table 3 Indexes in EVM.

指標（略称）	プロジェクトマネジメント	DBC アプリケーション
PV	実施すべき作業に割り当てられた予算（金額）	実施すべきタスクに割り当てられた金額もしくは電力
EV	完了した作業に相当する価値（金額）	完了したタスクに相当する金額もしくは電力
AC	ある時点までに投入したコストの総額（金額）	ある時点までに投入した金額もしくは電力の総額
SPI	完了した作業の時間の効率，EV/PV で算出される	完了したタスクの時間の効率，EV/PV で算出される
CPI	完了した作業の予算の効率，EV/AC で算出される	完了したタスクの予算の効率，EV/AC で算出される

3.2 EVM で用いられる指標

表 3 に示すとおり，EVM には代表的な指標が 5 つある．PV（Planned Value）が計画予算，EV（Earned Value）が進捗，AC（Actual Cost）が消化予算に相当する．PV は，実施すべき作業に割り当てられた予算（金額）を示す．EV は，ある時点までに完了した作業に相当する計画上の価値

表 4 EVM による分析例
Table 4 Examples of EVM analysis.

ケース	PV	EV	AC	SPI	CPI	分析結果
1	8 mJ	7 mJ	7 mJ	0.88	1.00	期限を守れない，総予算を守れる
2	8 mJ	5 mJ	7 mJ	0.63	0.71	期限を守れない，総予算を守れない

（金額）を示す．AC は，ある時点までに，作業実施のために投入した予算の総額（金額）を示す．SPI（Schedule Performance Index）は EV/PV で計算され，1 未満の場合には，完了した作業の時間の効率では期限を守れないことを示す．1 以上の場合には，完了した作業の時間の効率で期限を守れることを示す．CPI（Cost Performance Index）は，EV/AC で計算され，1 未満の場合には，完了した作業の予算の効率では総予算を守れないことを示す．1 以上の場合には，完了した作業の予算の効率で総予算を守れることを示す．プロジェクトマネジメントでは，SPI と CPI によりプロジェクトの状況を分析し，分析結果に基づいてプロジェクトへの変更を行う．

表 3 に示すとおり，プロジェクトマネジメントの EVM を，*DBC* アプリケーションにマッピングした．PV は，実施すべきタスクに割り当てられた電力もしくは金額を示す．EV は，ある時点までに完了したタスクに相当する計画上の電力もしくは金額を示す．AC は，ある時点までに，タスク実施のために投入した電力もしくは金額の総額を示す．SPI は EV/PV で算出され，1 未満の場合には，完了したタスクの時間の効率では期限を守れないことを示し，1 以上の場合には，完了したタスクの時間の効率で期限を守れることを示す．CPI は EV/AC で算出され，1 未満の場合には，完了したタスクの予算の効率では総予算を守れないことを示し，1 以上の場合には，完了したタスクの予算の効率で総予算を守れることを示す．*DBC* アプリケーションの EVM を表 2 の例にあてはめた結果を**表 4** に示す．表 4 に示すとおり，EVM では SPI と CPI により 2 つの状況を異なるものとして分析できる．

4. 提案手法

2.3 節であげた課題を解決するため，EVM を導入した適応フレームワークを提案する．本研究では，適応フレームワークを，「期限と総予算を順守するため，*DBC* アプリケーションの実行中に，状況を分析し，必要に応じて変更を加える適応ソフトウェアを構築するためのフレームワーク」と定義する．なお，適応ソフトウェアは，*DBC* アプリケーション外部のソフトウェアである．課題を解決するためには，実行中の *DBC* アプリケーションの状況を分析する必要がある．また，分析結果に応じて変更を加える必要もある．そこで，実行中の分析と変更を自動化する適応ソフトウェアが求められる．ソフトウェアが変化に適応するために自身の構造や振舞いを変更する自己適応技術 [8] において，適応ソフトウェアは保守性を向上させることから広く用いられている．以上のことから，課題解決策として，適応ソフトウェアを構築するための適応フレームワークを採用することは適当といえる．

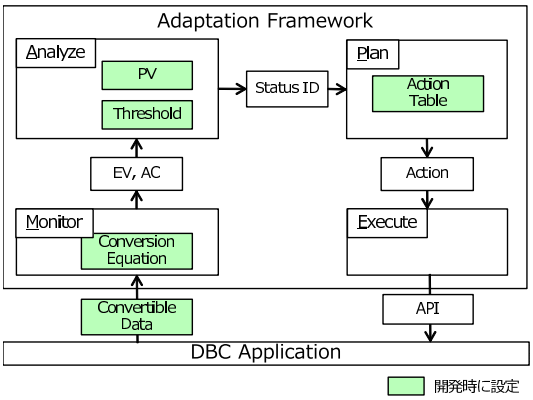

図 2 フレームワーク概要：MAPE ループを参考にした適応フレームワーク
Fig. 2 Overview of the adaptation framework: MAPE loop as reference.

自己適応技術では，適応ソフトウェアの構築に MAPE ループ [27], [28] を用いることが多い．MAPE ループは 4 つのコンポーネントから構成され，各コンポーネントは次の役割を持つ．*Monitor* は，アプリケーションや外部環境のデータを収集する．*Analyze* は，収集したデータから，アプリケーションが要求を満たすことができるか分析する．*Plan* は，アプリケーションが要求を満たせない場合に，変更方法を計画する．*Execute* は，計画された変更を実行する．我々は，**図 2** に示すとおり，MAPE ループを参考に適応フレームワークを構築した．本フレームワークは，実行時に *Monitor* が変換データを EV，AC に変換し，*Analyze* は，EV，AC，PV から SPI，CPI を計算し，それらをしきい値と比較することで *DBC* アプリケーションの状況を分析する．*Plan* は *StatusID* で与えられる状況の分析結果に基づき変更方法（以降，「アクション」）を選択し，*Execute* が，選択されたアクションに紐づく API を実行する．これらの処理を可能とするため，図 2 に示すとおり，開発時に PV，変換式，変換データ，しきい値，アクションテーブルを設定する．本章では，4.1 節でフレームワークの開発方法を記載し，これらの設定方法をモニタリングアプリケーションを例にとって示す．4.2 節でフレームワークの機能を説明する．

なお，本研究では，我々の提案に注力するため，以下の 3 つの前提状況を置く．

前提 1) DBC アプリケーションの存在

適応機能を持たない *DBC* アプリケーションはすでに存在すると仮定する．

前提 2) API

DBC アプリケーションの有する API を開発者は分析し，API の機能を把握できる．

前提 3) データの取得

DBC アプリケーションから，提案フレームワークは必要なデータを取得できる．

4.1 フレームワークの開発方法

4.1.1 PV の設定

PV は，*DBC* アプリケーションの実行期間中に利用する計算機器の電力もしくは金額を合算し，それに予備予算を加えることによって設定する．予備予算を設定する理由は，予算を要するアクションを実行する際の予算として充当するためである．たとえば，進捗が遅れている際に，ノードを追加し進捗を向上させる場合には予備予算が必要となる．

PV の設定を，モニタリングアプリケーションを例にとって説明する．データ A を取得するノードが 3 個，データ B を取得するノードが 1 個存在する．これらをまとめてソースと呼ぶ．また，データを保存するノードが 1 個存在する．これをシンクと呼ぶ．最後に，アプリケーションの実行状況を監視し分析の上変更を行うノードが 1 個存在する．これを PM（Project Manager）と呼ぶ．本アプリケーションは計 6 個のノードから構成される．時間 T の時点では，6 ノードを稼動させた電力分のタスクが達成されている計画となるため，PV は，式 (1) で与えられる．ここで，*CPU_idle* は CPU の待機時の電流，*Radio_Rx* は無線受信時の電流，V は電圧を示す[*1]．CPU の待機時の電流，無線受信時の電流のみを考慮している理由は，これらが本アプリケーションにおいて消費電力の 99%以上を占めるためである．なお，C は予備予算を含んだ比率であり，ここでは予備予算を 10%とし，1.1 とした．

$$PV = 6 * T * C * V * (CPU\_idle + Radio\_RX) \quad (1)$$

4.1.2 変換式の設定

AC の変換式は，利用する計算機器の実行時の金額もしくは電力を足し合わせることによって作成する．EV の変換式は，実行中の完了したタスクに相当する計画時の金額もしくは電力を算定することで作成する．

モニタリングアプリケーションにおいて，シンクやソース等の個々のノードの消費電力は，式 (2) で表現される．ここで，*T_P* はノードがアプリケーションに参加してからの経過時間，*T_RX* は受信待機時間，*T_TX* は送信時間，Radio_TX は送信電力[*2]，V は電圧を示す．複数のノードでアプリケーションが構成されるため，参加ノード数分だけ式 (2) を累計し，アプリケーション全体での消費電力を算出する，これが AC に相当する．AC は式 (3) で表現される．N はアプリケーションに参加したノード数を示す．

$$\begin{aligned} Cost\_Node = V * (&CPU\_idle * T\_P \\ &+ Radio\_RX * T\_RX \\ &+ Radio\_TX * T\_TX) \end{aligned} \quad (2)$$

$$AC = \sum_{i=1}^{N} Cost\_Node_i \quad (3)$$

EV は，シンクに収集されたデータ A およびデータ B のデータ数をもとに計上する．簡単には，それぞれ 1 つのデータを取得するのに必要な消費電力を，データ数分だけ計上したものである．EV は式 (4) で表現される．*N_Data_A* はシンクに収集されたデータ A の数，*N_Data_B* はシンクに収集されたデータ A の数を示す．*S_Int* はソースのサンプリング周期を示す．なお，*N_Data_A* に 4.5 および *N_Data_B* に 1.5 を乗じているのは，PM およびシンクの消費電力を 1.5 と 0.5 で配分しているためである．

$$\begin{aligned} EV = (&4.5 * N\_Data\_A + 1.5 * N\_Data\_B) * C * V \\ &* S\_Int * (CPU\_idle + Radio\_RX) \end{aligned} \quad (4)$$

なお，モニタリングアプリケーションにおける PV，EV，AC の整合性の確保については付録 A.1 に記載した．予備予算については，式 (1) と式 (4) に C を乗じていることから PV と EV の整合性は確保される．AC は，実行するアクションにより，PV 未満になる場合，PV 以上になる場合がある．

4.1.3 変換データの設定

変換データの設定では，変換式の構成要素のうち，適応フレームワークが *DBC* アプリケーションから取得すべきものを設定する．モニタリングアプリケーションにおいては，式 (2)，式 (4) の計算を可能とするため，AC については，*T_P*，*T_RX*，*T_TX* を取得するように設定し，EV については，*Num_Data_A*，*Num_Data_B* を取得するように設定する．

4.1.4 しきい値の設定

しきい値を設定することにより，分析される状況を振り分ける．本研究では，**表 5** に示すように状況を分析し，各状況にあわせてアクションを実行することを考える．たとえば，予算オーバの中にも，進捗遅れ，進捗どおり，進捗進みによってとるべきアクションを変えたい場合がある．そこで，SPI と CPI について，下のしきい値，上のしきい値を設定する．SPI，CPI の下のしきい値 th_{lower} には，完

[*1] 本研究では WSN のノードとして MICAz を想定した．文献 [22] にならい CPU_idle を 4.93 mA，Radio_Rx を 19.70 mA とした．V は 3.0 V となる．

[*2] 文献 [22] にならい Radio_TX を 17.40 mA とした．

表 5 Status ID と分析される状況
Table 5 Status IDs and analyzed statuses.

	CPI<th_{lower}	th_{lower} ≤CPI≤th_{upper}	th_{upper}<CPI
SPI<th_{lower}	(11) 進捗遅れ・予算オーバ	(21) 進捗遅れ・予算どおり	(31) 進捗遅れ・予算内
th_{lower} ≤SPI≤th_{upper}	(12) 進捗どおり・予算オーバ	(22) 進捗どおり・予算どおり	(32) 進捗どおり・予算内
th_{upper}<SPI	(13) 進捗進み・予算オーバ	(23) 進捗進み・予算どおり	(33) 進捗進み・予算内

了時に達成したい SPI，CPI を設定する．通常，*DBC* アプリケーションでは，完了時にタスクを 100%完了しており，総予算を 100%順守することが必要であるため，双方ともに 1.0 を設定する．次に，SPI の上のしきい値 th_{upper} には，実行中の SPI の上限を設定する．*DBC* アプリケーションにおいて，タスクが早く完了することは望ましいが，実行中の SPI が 1 を大きく超える場合には，計画が妥当でない場合も含まれることに留意する必要がある．CPI の上のしきい値 th_{upper} には，実行中の CPI の上限を設定する．同様に，*DBC* アプリケーションにおいて，予算を順守することは望ましいが，CPI が 1 を大きく超える場合には，計画が妥当でない場合も含まれる．これらの設定値は，*DBC* アプリケーションによって異なるが，設定例としては 1.05 や 1.10 となる．

モニタリングアプリケーションでは，SPI の th_{lower} として 0.90，CPI の th_{lower} として 1.00 を設定した．SPI の th_{lower} として 0.90 を設定している理由は，本アプリケーションでは，パケットロスが発生し，取得できなかったデータと同時刻のデータを後から取得はできないことから，少なくとも 90%以上のデータを取得するとしているためである．また，本アプリケーションでは，未来のデータを取得することもできないため，SPI の th_{upper} には，理論上の最大値である 1.00 を設定した．CPI の th_{upper} には 1.05 を設定した．

4.1.5 アクションテーブルの設定

アクションテーブルの設定では，*DBC* アプリケーションの API を分析し，表 5 に示す状況を改善する API をアクションとしてテーブルに設定する．ただし，提案手法では，StatusID が 22，23，32，33 の場合は問題がない状況として，アクションは設定しない．

アクションテーブルの設定は次の考え方を基本とする．現在から次の分析時刻までの PV, EV, AC の差分を ΔPV，ΔEV，ΔAC とおく．StatusID が 12 の場合には，CPI を向上させるため，$\Delta EV/\Delta AC > 1$ となるアクションを設定する．StatusID が 21 の場合には，SPI を向上させるため，$\Delta EV/\Delta PV > 1$ となるアクションを設定する．StatusID が 11 の場合には，SPI，CPI 双方を向上させるため，$\Delta EV/\Delta PV > 1$，$\Delta EV/\Delta AC > 1$ となるアクションを設定する．StatusID12 および StatusID 21 のアクション双方を設定するか，それが困難な場合には，どちらかを優先させて片方のアクションを設定する．StatusID が 13 の場合には，SPI を低下させても CPI を向上させたいため，$\Delta EV/\Delta PV < 1$，$\Delta EV/\Delta AC > 1$ となるアクションを設定する．StatusID が 31 の場合には，CPI を低下させても SPI を向上させたいため，$\Delta EV/\Delta PV > 1$，$\Delta EV/\Delta AC < 1$ となるアクションを設定する．なお，設定するアクションは，*DBC* アプリケーションに依存するため，この考え方に従って設定できるとは限らない．

表 6 モニタリングアプリケーションのアクションテーブル
Table 6 Action table in monitoring application.

StatusID	アクション
11	$\Delta EV/\Delta AC > 1$ となるよう，参加しているノードの無線を一定期間，定期的にスリープさせる（SPI の改善よりも CPI の改善を優先する）．
12	$\Delta EV/\Delta AC > 1$ となるよう，参加しているノードの無線を一定期間，定期的にスリープさせる．
13	$\Delta EV/\Delta PV < 1$，$\Delta EV/\Delta AC > 1$ となるよう，単位時間あたりの AC が最も大きいソースを離脱させる．
21	$\Delta EV/\Delta PV > 1$ となるよう，計画よりもソース数が少ない場合には，ソースを追加する．ただし，ソース数が計画以上の場合には，EV の獲得効率が最も悪いソースが進捗遅れの原因とし，そのソースを新たなソースに変更する．
31	$\Delta EV/\Delta PV > 1$，$\Delta EV/\Delta AC < 1$ となるよう，ソースを追加する．ただし，直近の EV の ΔEV が一定の数値以上である場合には，ソースを追加しない．

モニタリングアプリケーションでは，API として，ソースおよびシンクを追加，変更，離脱させる API，一定期間参加ノードの無線をスリープさせることにより消化予算を削減する API が存在した．そこで，我々は**表 6** に示すとおり，アクションテーブルを作成した．たとえば，StatusID が 11 の際は，進捗遅れ・予算オーバであり，ここでは進捗遅れを改善するよりも，予算オーバを改善することを優先し，参加しているノードの無線を一定期間，定期的にスリープさせるアクションを設定した．

4.2 フレームワークの機能

我々の提案するフレームワークを**図 3** に示す．以下にフレームワーク実行時の各コンポーネントの機能を記載する．

Monitor：*DBC* アプリケーションから変換データを取得し，変換データを変換式により EV と AC に変換する．変換した EV と AC を *Analyze* に報告する．この処

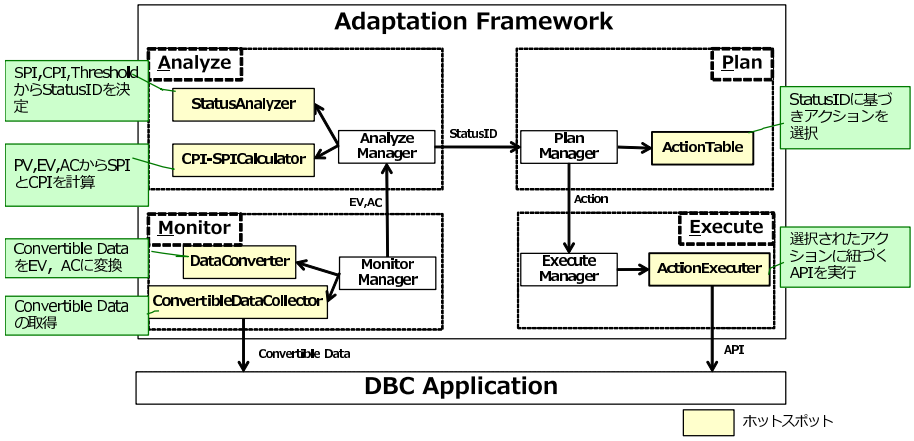

図 3 提案する適応フレームワークの機能とホットスポット
Fig. 3 Functionalities and hot spots of proposed adaptation framework.

理は，*MonitorManager* が *ConvertibleDataCollector* および *DataConverter* を利用して実行する．*ConvertibleDataCollector* は，*DBC* アプリケーションの API を利用することによって変換データを取得する．*DataConverter* は，取得された変換データを，変換式によって EV と AC に変更する．

Analyze：*Monitor* から報告される EV，AC，設定されている PV から SPI と CPI を計算し，これらをしきい値と比較することにより状況を分析する．具体的には，表 5 に示す StatusID を出力し，*Plan* に報告する．この処理は，*AnalyzeManager* が，*CPI-SPICalculator* および *StatusAnalyzer* を利用して実行する．*CPI-SPICalculator* は，4.1 節で述べた EV，AC，PV から SPI と CPI を計算する．*StatusAnalyzer* は，*DBC* アプリケーションの実行状況を，SPI，CPI，しきい値から，表 5 に基づき分析し，StatusID を出力する．

Plan：*Analyze* から報告される StatusID と設定されたアクションテーブルに基づき変更要否を決定し，アクションテーブルからアクションを選択し，アクション名を *Execute* に報告する．この処理は，*PlanManager* が *ActionTable* を利用して実行する．*ActionTable*は，*AnalyzeManager* から報告される StatusID を受け取り，アクションテーブルと StatusID をマッチングし，変更が必要な場合には，アクションテーブルから状況に即したアクションを選択する．

Execute：*Plan* から報告されるアクション名に紐づく API を実行する．この処理は，*ExecuteManager* が *ActionExecutor*を利用して実行する．*ActionExecuter* は，*DBC* アプリケーションの API とアクション名の関連を管理し，*PlanManager* から報告されるアクション名に紐づく API を呼び出す．

図 3 に示すとおり，本フレームワークには，*ConvertibleDataCollector*，*DataConverter*，*CPI-SPICalculator*，*StatusAnalyzer*，*ActionTable*，*ActionExecuter* の計 6 つのホットスポットがある．*DBC* アプリケーションに応じて，これらのコンポーネントを変更する必要がある．

5. 評価

本章では，モニタリングアプリケーションを対象に実装した適応フレームワークの評価結果を示す．5.1 節において，実装したフレームワークおよび評価環境を述べる．5.2 節で，EVM を実装した提案フレームワークと個別指標を実装したフレームワークのパフォーマンスをシミュレーション環境で比較する．5.3 節では，提案フレームワークの有効範囲を評価する．5.4 節で，提案フレームワークのオーバヘッドを述べる．

5.1 実装および評価環境

モニタリングアプリケーションは，TinyOS 2.1.2 [17] 上に NesC 1.3.1-1 [11] により実装されていた．そこで，適応フレームワークも同様の環境で構築した．構築したフレームワークを図 **4** に示す．4.2 節で述べたホットスポットにならい，*WSNDataConverter*，*WSNConvertibleDataCollector*，*WSNStatusAnalyzer*，*WSNCPI-SPICalculator*，*WSNActionTable*，*WSNActionExecuter* の 6 つのコンポーネントを構築した．モニタリングアプリケーションは，次の 3 つのモジュールから構成される．

SenseM：ソースでの計測，シンクへのデータの集計，保存を行う．モニタリングアプリケーションの変換データである *Num_Data_A*，*Num_Data_B* を *WSNConvertibleDataCollector* に報告する．また，*WSNActionExecuter* がノードを一定期間スリープさせる API を実行した際には，当該処理を実行する．

EnergyMgrM：各ノードでの消化予算に関する情報を管理する．モニタリングアプリケーションの変換データである *T_P*，*T_RX*，*T_TX* を *WSNConvertibleDataCollector* に報告する．

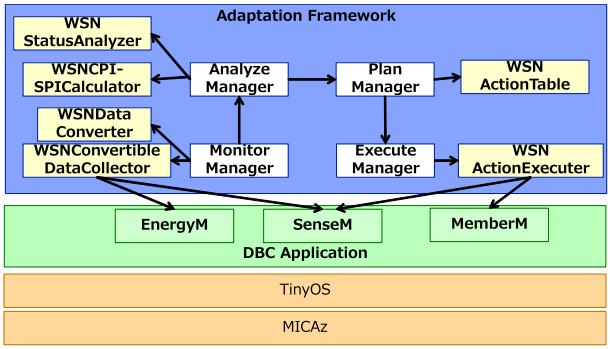

図 4 モニタリングアプリケーションで実装した適応フレームワーク
Fig. 4 Framework implemented for the monitoring application.

表 7 シミュレーション環境
Table 7 Simulation environment.

項目	説明
トポロジ	グリッド（4 × 4）
ノード間隔	1.5 m
シミュレーション時間	2,500 秒
サンプリング周期	2 秒
データ送信周期	10 秒
定期報告・分析の周期（Slot）	50 秒
総予算	1,220 J

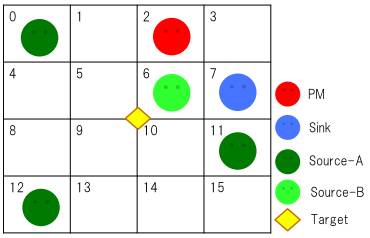

図 5 ソース，シンク，PM の初期配置
Fig. 5 Initial deployment of sources, sink and PM.

MemberM：アプリケーションを構成するノードを管理する．PM，シンク，ソース（データ A およびデータ B）の位置や，各ノードの参加時間等を管理する．また，*WSNActionExecuter* がノードを追加，変更，離脱させる API を実行した際には，当該処理を実行する．

構築したフレームワークを，シミュレーション環境 [16] において評価した．計測条件を**表 7** に示す．なお，サンプリング周期はソースが計測を行う周期，データ送信周期は，ソースからシンクへのデータの送信周期である．定期報告・分析の周期は，ソースおよびシンクが PM ノードに変換データを報告する周期，PM が分析を行う周期を示す．ここでは，便宜上 Slot と呼ぶ．**図 5** に示すとおり，4 × 4 のグリッドのネットワークをシミュレーション環境で構築した．図 5 における数値はノード ID を示す．シミュレーション開始直後の PM，シンク，ソース A，ソース B の配置は，図 5 に示すノードに配置した．なお，ターゲットは，観測対象を示す．アクションの実行により，ソースを変更・追加する場合がある．ソース A を変更・追加する場合には，ノード 1，3，4，8，14，15 から選択する[*3]．たとえばソース A を変更する場合，ノード 0 からノード 1 に変更する．ソース B を変更・追加する場合には，ノード 5，9，10 からランダムに選択する．たとえばソース B を追加する場合，ノード 5 を追加する．

表 8 評価で用いた指標
Table 8 Indexes used in evaluation.

手法	進捗の指標	予算の指標
個別指標	SPI（EV/PV）	PV/AC
提案手法	SPI（EV/PV）	CPI（EV/AC）

表 9 提案手法と個別指標の比較
Table 9 Comparison with conventional approach.

手法	EV (J)	AC (J)	SPI	CPI	PV/AC	超過数
個別指標	1,149	1,206	0.94	(0.95)	1.01	9
提案手法	1,117	1,099	0.92	1.02	-	0

5.2 個別指標との比較

EVM と個別指標の比較を行うために双方を実装したフレームワークを準備した．EVM と個別指標の大きな差異は，CPI により進捗にあわせて予算を消化する点にある．この効果に着目するため，**表 8** に示すとおり，個別指標の進捗の指標については，提案手法と同様に SPI を採用した．一方，予算の指標としては，PV/AC を用いた．なお，提案手法と個別指標を実装したフレームワークで，しきい値，StatusID，アクションに相違はない．

提案手法と個別指標で，予備予算を 10%としたときのシミレーションを 50 回数実施した際の結果を，**表 9** に示す．提案手法，個別手法ともに期限内に進捗は達成することができた．ただし，個別指標では，総予算（1,220 J）を超過してしまう場合が 9 回計測された．一方で，提案手法は 50 回すべてで総予算の制約を守った．個別指標における総予算の超過は，EV と関係しない PV/AC を分析することに原因がある．個別指標では，EV に依存せず AC を消化する．PV/AC が期限間際に下のしきい値を下回り，AC を削減する変更が間に合わない場合に総予算の超過が発生する．これに対して，提案手法は CPI を分析するため，EV に見合う AC を消費するように適応する．これが，提案手法の 1,099 J，個別指標の 1,206 J という AC の相違に現れている．また，1 回のシミュレーションでは 50 回の分析を行うが，個別指標と提案手法は，1 回のシミュレーションあたり平均して 20.1 回の異なる StatusID を出力していた．これらのことから，提案手法は *DBC* アプリケーションの分析方法として既存手法より適しているといえる．

[*3] 複数のソース A で囲まれる面積が最大になるようにノードを選択する簡易なアルゴリズムを採用している．ノードの選択手法は本研究の対象外となる．

表 10 SPI の下のしきい値を変化させた場合の AC
Table 10 AC with different SPI lower threshold.

手法	0.8	0.85	0.90
個別指標	1,136	1,170	1,206
提案手法	1,030	1,066	1,099
個別指標/提案手法	90.6%	91.1%	91.1%

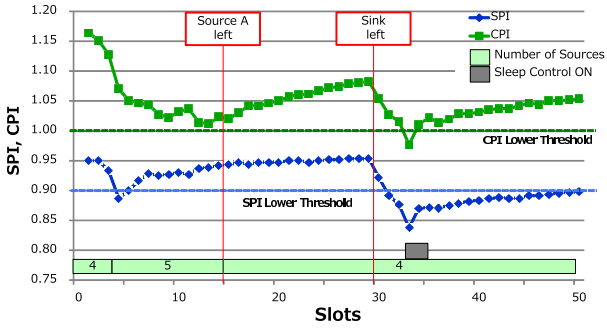

図 6 適応時の SPI と CPI の変化
Fig. 6 Variation in SPI and CPI during adaptation.

なお，AC の違いを確認するために，SPI の下のしきい値を 0.8，0.85 とした場合の AC を評価した．**表 10** は 30 回の平均値を示す．提案手法は，おおむね 10%程度消費電力を節約できることが分かる．

5.3 適応範囲の評価

提案フレームワークの適応範囲を確認するために，Slot 15 でソース A が離脱し，Slot 30 でシンクが離脱するシナリオを評価した．予備予算が 20%として，シミュレーションを 30 回実行した際の代表的な例を**図 6** に示す．図 6 を見ると，Slot 5 の段階では，パケットロスの影響から進捗がのびず，SPI の下のしきい値である 0.90 を下回っている．そこで，ソース A を追加する適応を行い，計 5 個のソースで進捗を回復している．その後，Slot 15 でソース A が離脱した際は，ソース数は計 4 個となるが，SPI の下のしきい値を満たしていることから変更は実施しない．その後，Slot 30 でシンクが離脱した後，Slot 34，Slot 35 では，無線を一定期間スリープさせる変更を行い，CPI を回復させている．このように，本フレームワークは変化に適応する．

本フレームワークの適応範囲は，予備予算の比率によって変化する．これは，PV に余裕がある方が，とりうる策が豊富なためである．モニタリングアプリケーションのアクションテーブルでは，予備予算が大きく，PV に余裕があればソースの追加等の計算機器の投入が可能になり，EV が向上する．この結果，CPI が改善される．一方，予備予算が十分でない場合は，AC を削減する適応を優先し，EV も向上しない．この結果，CPI の改善ペースも落ちる．**表 11** に予備予算を 10%，20%，30%と変化させた場合の評価結果（20 回の平均値）を示す．表 11 より，予備予算が

表 11 予備予算を変化させた場合の SPI と CPI
Table 11 SPI and CPI obtained in different reserve rate.

予備予算	EV (J)	AC (J)	総予算 (J)	SPI	CPI
10%	832	938	1,220	0.68	0.89
20%	1,192	1,129	1,320	0.90	1.06
30%	1,342	1,208	1,440	0.93	1.08

10%の際には，適応範囲が限られ，状況の悪化を止められないことが分かる．変化の発生頻度や影響の大きさといったリスクにあわせて，適した予備予算を設定する方法は課題となる．

5.4 オーバヘッド

DBC アプリケーション単体のコード容量，*DBC* アプリケーションと本フレームワーク合計のコード容量は，63 Kbyte，81 Kbyte となった．フレームワーク分のオーバヘッドは 18 Kbyte となる．今回想定した MICAz のプログラムメモリは 128 Kbyte であり，そのオーバヘッドはプログラムメモリの 14%となり無視できるとはいいがたい．適応性の獲得とコード容量とのトレードオフを考慮して決定することが必要となる．

6. 議論

6.1 提案手法の限界

今回提案した適応フレームワークには次の限界がある．

6.1.1 EVM の性質に依存する限界

EVM は計画（PV）と実績（AC，EV）を比較することによって，状況を分析する手法である．このため，PV を作成できないアプリケーションには適用することができない．たとえば，実行中に確定するユーザの位置を予測し，予測された位置にあわせて計算機器の設定を前もって変更するような例 [26] では，実行前に PV を作成することができないことから提案手法は利用できない．

EVM は進捗と予算を管理するための手法であり，品質管理は間接的となる．本研究における *DBC* アプリケーションでは，計画されたタスクが実行されれば，要求品質を満たすという前提を置いている．要求品質を満たしたか否かは，EVM においても間接的に管理される．ただし，実行したタスクの品質を直接管理したい場合には，他手法との併用が必要になる．モニタリングアプリケーションの場合には，データを取得できれば要求品質を満たすとしている．たとえば，センサデータにエラーが含まれる場合には，エラーを検知・分類する手法 [6] と併用してエラーを取り除く必要がある．

なお，AC に電力と金額が混在するような場合には，それぞれの単位で EVM を個別に管理する必要が生じ，提案手法をそのまま適用することはできない．たとえば，自分で所有する計算機器と，インフラ事業者の提供する計算機器

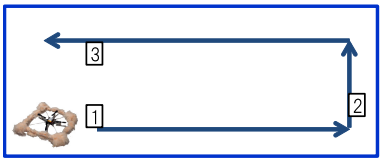

図 7 掃除アプリケーションの例
Fig. 7 Example of cleaning application.

を双方を用いて，タスクを実行するような場合が相当する．

6.1.2 適応の限界

本研究では，*DBC* アプリケーションの API を分析し，それに基づき変更可能箇所をアクションテーブルに登録する手法をとった．そのため，適応の限界は *DBC* アプリケーションの変更可能範囲となる．仮に，これらの変更可能範囲のアクションで，変化に対応できない場合には，提案手法を用いても変化に適応することはできない．

6.1.3 Plan の限界

提案フレームワークの *Plan* は，アクションテーブルから，StatusID に基づきアクションを選択する単純な処理を行う．同じ StatusID において，実行したアクションのフィードバックを用いて異なるアクションを選択することや，アクションの実行順序を考慮して複数のアクションを実行する等の複雑な処理は実行していない．*Plan* の機能の充実は今後の課題となる．

6.2 適用例

提案フレームワークを，異なる *DBC* アプリケーションに適用した例を示す．UAV (Unmanned Aerial Vehicle) [2] にモップを取り付け，UAV を**図 7** に示すパスに沿って移動させながら，モップや UAV のロータの引き起こす風で机の上を掃除する *DBC* アプリケーションを考える [25]．本 *DBC* アプリケーションの期限は 30 秒，総予算は 7.5 J とする[*4]．

詳細は付録 A.2 に記載するが，4.1 節で述べた手順に従い PV，変換データ，変換式，しきい値，アクションテーブルを設定し，**図 8** に示す UAV 適応フレームワークを Python 2.7 により実装した．UAV の *DBC* アプリケーションの LOC は 981 行であり，適応フレームワークの LOC は 182 行となった．提案フレームワークを利用することで，工数のオーバヘッドが少なく開発することができた．また，UAV 適応フレームワークを実機で評価し，人手で UAV を遮る等進捗を遅らせても，飛行モードを変更して，期限と総予算の制約を守る適応が見られた．提案フレームワークは，*DBC* アプリケーションの制約を守るための 1 つのアプローチになると考える．

[*4] UAV として用いた Crazyflie は 1 秒飛行するために 0.415 J 消費する．計画では 30 秒のうち 15 秒間飛行を行い，移動や掃除を行う．予備予算を 20%に設定したため，$0.415 * 15 * 1.2$ で，7.5 J を総予算に設定した．

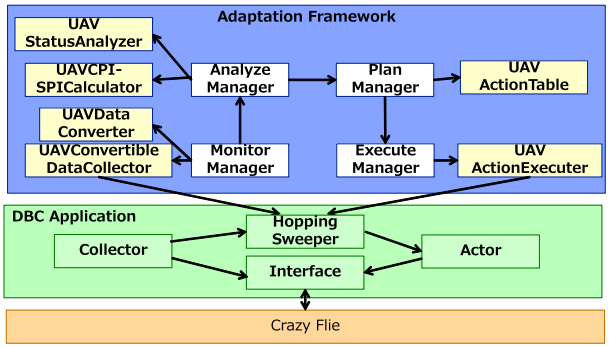

図 8 掃除アプリケーションで実装した適応フレームワーク
Fig. 8 Framework implemented for the cleaning application.

6.3 EAC の利用可能性

DBC アプリケーションでは，期限内にタスクが完了すれば，総予算の範囲内での予算消化は可能である．表 9 および表 10 を見ると，提案手法では 1,220 J の総予算に対し余裕を残していることが分かる．EVM には，実行中に完了時の消化予算を推定する指標として EAC (Estimated At Completion) がある．EAC は，$EAC = AC + (Total\_Budget - EV)/CPI$ で計算される．表 11 の例にあてはめると，EAC は上から 1,374 J，1,250 J，1,299 J となる．予備予算が 20%と 30%の場合，実行中に総予算が超えないことを計算できるため，実行の終盤では，SPI，CPI に加えて EAC を分析に加えることで，総予算を有効に活用するといった改善は可能と考える．

7. 関連研究

クラウドコンピューティングや，グリッドコンピューティングにおいて，総予算や期限の制約を取り扱い，タスクをこなすスケジューリングアルゴリズムが多く研究されてきた．スケジューリングアルゴリズムには，大きく，実行前にタスクの情報が得られている場合を想定したオフラインスケジューリング，実行中にタスクの情報が得られることを想定したオンラインスケジューリングに分類される．前者の例に，数値計算等がワークフローで与えられ，実行前にワークフローの処理をスケジューリングする手法 [18], [23] がある．後者の例に，実行時に投入されるタスクを期限内に最小のコストでスケジューリングする手法 [19], [20] がある．本研究では，実行前にタスクの情報が得られる環境を想定しているため，提案手法における PV の作成にオフラインスケジューリングの研究は参考にできる．ただし，オフラインスケジューリングは一般に実行中の変化を考慮していないため，変化に適応する際には，リスケジュールが必要になる．期限の制約を守るために実行中の状況を分析し，リスケジューリングを行う手法は，複数提案されている．文献 [31] では，実績時間と計画時間の比としきい値を比較することで，リスケジュールを行う方法を提案している．同様に，文献 [29] では，実績時間が計

画時間を上回った場合，リスケジュールの要否を決定する方法を提案している．文献 [30] においても，実績時間をモニタリングし，計画時間とのパフォーマンスの差が大きい場合にはリスケジュールを行う手法を提案している．これらの研究では，期限の制約を対象とし，消化予算を分析して変更することは対象としていない点，変更がタスクのリスケジュールに限られている点が，提案手法と異なる．

サービスコンピューティングと自己適応の研究分野では，制約を守るために，予測に基づく適応が多く提案されている．たとえば，文献 [5] では，ビジネスプロセスの実行中の QoS をモニタリングし，実行中の QoS が計画値を上回った場合には，集約関数 [24] を用いて SLA（QoS の累計）を予測し，SLA を維持できない場合には適応を行う手法を提案している．文献 [21] では，ワークフローを構成するサービスの応答時間等の過去データを用い，オンラインテストでこれらのデータの予測を行い，予測値としきい値を比較分析することで，適応を行う手法を提案している．また，文献 [15] では，実行時のデータのモニタリングと，機械学習により，SLA が維持可能であるか分析し，維持できない場合には適応を行う手法を提案している．これらの予測に基づく手法は，過去データの活用や，アプリケーションに特化した予測方法を用いており，過去データがない場合や，ドメインの異なるアプリケーションに適用したい場合には適用することが難しい．本研究で提案する EVM は，PV を作成し，EV および AC を取得できるアプリケーションであれば汎用的に利用できる点が異なる．

EVM の前身は 1960 年代に米国の DoD で利用され始め，1990 年代後半に民間向けに改訂され，EVM と呼ばれていくことになった [10]．現在，EVM の有効性については広く知られており，多くの企業で利用されている．EVM に関する研究には，プロジェクトコストの過去データを用いることで EVM の指標の精度を向上させる手法の提案 [9]，EAC の改善提案 [14] 等がある．これらの研究は，EVM の指標そのものに関する研究であり，本研究で想定するような適応を前提として EVM を利用するものとは異なる．

8. まとめ

本研究では，予算と時間の制約を満たしつつ，タスクを完了するタイプのアプリケーションを想定した．開発時に詳細までは把握できない変化が生じると，アプリケーションの制約が守られない場合がある．本研究では，変化に適応し制約を守るための適応フレームワークを提案した．変化に適応するためには，状況を正しく分析することが重要である．我々は，提案フレームワークに，プロジェクトマネジメントの進捗管理と予算管理で用いられる EVM（Earned Value Management）による分析手法を実装した．シミュレーション環境において，既存手法と比較を行い，提案フレームワークが既存手法に比べてアプリケーションの制約を守る可能性を向上させたことを示した．将来研究として，進捗・予算に品質を加えた分析・適応方法，提案フレームワークのオートスケーリングへの適用を行う．本研究が，変化に適応する際の手法の 1 つになれば幸いである．

謝辞 本研究の実施，執筆にあたり有益な助言をしていただいた本位田真一教授（国立情報学研究所）に感謝する．

参考文献

[1] Amazon Auto Scaling (online), available from ⟨http://aws.amazon.com/jp/autoscaling/⟩.
[2] Crazyflie Wiki (online), available from ⟨http://wiki.bitcraze.se/projects:crazyflie:index⟩.
[3] RightScale (online), available from ⟨http://www.rightscale.com/⟩.
[4] *A Guide to the Project Management Body of Knowledge* (*PMBOK Guide*) – *5th Edition*, Project Management Institute (2013).
[5] Aschoff, R.R. and Zisman, A.: Proactive adaptation of service composition, *2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (*SEAMS*), pp.1–10, IEEE (2012).
[6] Baljak, V., Kenji, T. and Honiden, S.: Faults in Sensory Readings: Classification and Model Learning, *Sensors & Transducers*, Vol.18, pp.177–187 (2013).
[7] Buyya, R. and Murshed, M.: A deadline and budget constrained cost-time optimisation algorithm for scheduling task farming applications on global grids, *arXiv preprint cs/0203020* (2002).
[8] Cheng et al.: Software Engineering for Self-Adaptive Systems: A Research Roadmap, *Software Engineering for Self-Adaptive Systems*, Lecture Notes in Computer Science, Vol.5525, pp.1–26, Springer (online), available from ⟨http://dblp.uni-trier.de/db/conf/dagstuhl/adaptive2009.html⟩ (2009).
[9] de Souza, A.D.: A proposal for the improvement of project's cost predictability using EVM and historical data of cost, *ICSE*, pp.1447–1449 (2013).
[10] Fleming, Q.W. and Koppelman, J.M.: Earned value project management, Project Management Institute Pennsylvania (2000).
[11] Gay, D. et al.: The nesC language: A holistic approach to networked embedded systems, *PLDI*, pp.1–11 (2003).
[12] Gungor, V.C., Lu, B. and Hancke, G.P.: Opportunities and challenges of wireless sensor networks in smart grid, *IEEE Trans. Industrial Electronics*, Vol.57, No.10, pp.3557–3564 (2010).
[13] Iosup, A., Yigitbasi, N. and Epema, D.: On the performance variability of production cloud services, *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (*CCGrid*), pp.104–113, IEEE (2011).
[14] Iranmanesh, H., Mojir, N. and Kimiagari, S.: A new formula to "Estimate At Completion" of a Project's time to improve "Earned value management system", *2007 IEEE International Conference on Industrial Engineering and Engineering Management*, pp.1014–1017, IEEE (2007).
[15] Leitner, P. et al.: Monitoring, prediction and prevention of sla violations in composite services, *2010 IEEE International Conference on Web Services* (*ICWS*), pp.369–376, IEEE (2010).
[16] Levis, P. et al.: TOSSIM: Accurate and scalable simu-

lation of entire tinyOS applications, *SenSys '03: Proc. 1st International Conference on Embedded Networked Sensor Systems*, pp.126–137, ACM Press (online), DOI: http://doi.acm.org/10.1145/958491.958506 (2003).
[17] Levis, P. et al.: TinyOS: An operating system for sensor networks, *Ambient Intelligence*, pp.115–148, Springer (2005).
[18] Malawski, M. et al.: Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, *Proc. International Conference on High Performance Computing, Networking, Storage and Analysis*, p.22, IEEE Computer Society Press (2012).
[19] Mao, M. and Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows, *Proc. 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, p.49, ACM (2011).
[20] Mao, M., Li, J. and Humphrey, M.: Cloud auto-scaling with deadline and budget constraints, *2010 11th IEEE/ACM International Conference on Grid Computing* (*GRID*), pp.41–48, IEEE (2010).
[21] Metzger, A. et al.: Towards pro-active adaptation with confidence: Augmenting service monitoring with online testing, *Proc. 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp.20–28, ACM (2010).
[22] Perla, E. et al.: PowerTOSSIM z: realistic energy modelling for wireless sensor network environments, *Proc. 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, *PM2HW2N '08*, pp.35–42, ACM (online), DOI: 10.1145/1454630.1454636 (2008).
[23] Sakellariou, R. et al.: Scheduling workflows with budget constraints, *Integrated Research in GRID Computing*, pp.189–202, Springer (2007).
[24] Schuller, D. et al.: Optimization of complex qos-aware service compositions, *Service-Oriented Computing*, pp.452–466, Springer (2011).
[25] Tei, K., Aizawa, K., Suenaga, S., Takahashi, R., Lee, S. and Fukazawa, Y.: HoppingDuster: self-adaptive cleaning robot based on aerial vehicle, *Proc. 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pp.271–274, ACM (2014).
[26] Vansyckel, S. et al.: Configuration Management for Proactive Adaptation in Pervasive Environments, *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems* (*SASO*), pp.131–140, IEEE (2013).
[27] Vromant, P. et al.: On interacting control loops in self-adaptive systems, *Proc. 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp.202–207, ACM (2011).
[28] Weyns, D. et al.: On patterns for decentralized control in self-adaptive systems, *Software Engineering for Self-Adaptive Systems II*, pp.76–107, Springer (2013).
[29] Yu, J. et al.: Cost-based scheduling of scientific workflow applications on utility grids, *1st International Conference on e-Science and Grid Computing*, p.8, IEEE (2005).
[30] Yu, Z. and Shi, W.: An adaptive rescheduling strategy for grid workflow applications, *Parallel and Distributed Processing Symposium, IPDPS 2007, IEEE International*, pp.1–8, IEEE (2007).
[31] Zhang, Y. et al.: Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid, *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'09*, pp.116–123, IEEE (2009).

付　　録

A.1 PV，EV，AC の整合性

A.1.1 PV と EV

式 (4) において，ロスなくデータが取得できた場合，サンプリング周期が同じであるデータ A とデータ B の数は同一になる．データ A とデータ B の数を Num とおくと，式 (4) は式 (A.1) となる．ここで，データ数 Num とサンプリング周期 S_Int の積は T となる．以上のことから，PV を示す式 (1) と EV を示す式 (4) の整合性は保たれる．

$$EV = 6 * Num * S\_Int * C * V * (CPU\_idle + Radio\_RX) \quad \text{(A.1)}$$

A.1.2 PV と AC

アクションを実行しない場合には，特定の 6 個のノードでアプリケーションが実行される．この場合，式 (3) は式 (A.2) で示される．

$$AC = 6 * V * T * (CPU\_idle + Radio\_RX) + 6 * V * Radio\_TX * T\_TX \quad \text{(A.2)}$$

式 (3) における T_P，T_RX が，式 (A.2) において T になっている理由は，各ノードは T 時間参加し，T 時間受信待機するためである．PV の式 (1) と，式 (A.2) の第 1 項は，予備予算 C が 1.0 の場合は同一になる．また，モニタリングアプリケーションにおける式 (A.2) の第 2 項の影響は，**表 A·1** に示すとおり，0.02%と無視できるほど小さい．アクションがなく，C が 1.0 の場合には，PV を示す式 (1) と AC を示す式 (3) の整合性は保たれる．なお，表 A·1 は，ソース A を理想的なネットワーク環境で稼動させた際のサンプルデータを示す．

A.2 UAV 掃除アプリケーションの設定情報

UAV 掃除アプリケーションにおいて，PV，変換式，変換データ，しきい値，アクションテーブルを次のとおり設定した．PV は式 (A.3) で与えた．C は予備予算，T は実行時間を示す．EV の変換式は式 (A.4) で与えた．pl_actual

表 A·1 ノードの消費電力の内訳
Table A·1 Detail of energy consumption.

項目	数式	消費電力 (J)	割合 (%)
CPU アイドル	*T*V*CPU_idle*	36,960	20.01
受信待機	*T*V*Radio_RX*	147,690	79.97
送信	*T_TX*V*Radio_RX*	37	0.02
合計		184,687	100

表 A·2 UAV 掃除アプリケーションのアクションテーブル
Table A·2 Action table in UAV cleaning application.

Status ID	アクション
11	SPI の改善を優先し，$\Delta EV/\Delta PV > 1$ となるよう，飛行時間を増加し，ロータの吹き起こした風で掃除を行うモードを 0.8 秒間行う．
12	$\Delta EV/\Delta AC > 1$ となるような適切な API が存在しないため，変更を行わず，通常モードでの移動を維持する．
13	$\Delta EV/\Delta PV < 1$ となるよう，0.2 秒間着地させた状態を維持する．
21	$\Delta EV/\Delta PV > 1$ となるよう，飛行時間を増加し，ロータの吹き起こした風で掃除を行うモードを 0.8 秒間行う．
31	$\Delta EV/\Delta PV > 1$，$\Delta EV/\Delta AC < 1$ となるよう，飛行時間を増加し，ロータの吹き起こした風で掃除を行うモードを 1.2 秒間行う．

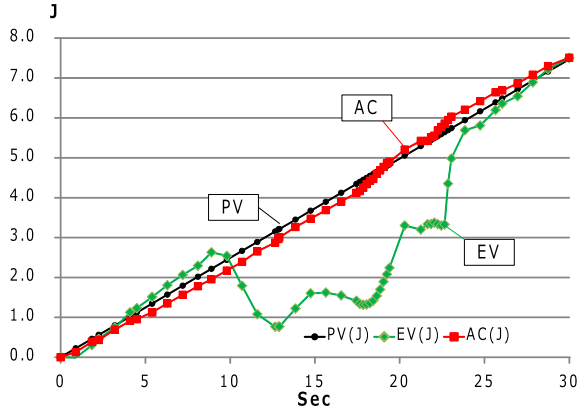

図 A·1 掃除アプリケーションの適応例
Fig. A·1 Adaptation example of cleaning application.

はパス上の現在の位置を示し，実行中の UAV の二次元座標から特定される．*pl_planned* は計画時のパスの総長を示す．また，AC の変換式は，式 (A.5) で与えた．*T_f* は飛行時間を示す．変換データは，実行中の UAV の二次元座標，飛行時間とした．しきい値は，SPI，CPI ともに，th_{lower} として 1.00，th_{upper} として 1.10 を設定した．アクションテーブルは，**表 A·2** のとおり設定した．表 A·2 における通常モードは 0.4 秒間低く飛行することで UAV に取り付けられたモップで机上を掃除するモードを示す．なお，すべての飛行モードで計画したパスからずれが生じた場合に修正を行う．**図 A·1** に示すとおり，10 秒の時点で，人手により UAV をパスに沿って後戻りさせる．これにより，EV は低下する（10 秒〜13 秒）．UAV は，飛行モードを変更し，EV を向上させる適応により（13 秒〜26 秒），期限と総予算の制約を守った（30 秒）．

$$PV = 0.415/2 * C * T \quad \text{(A.3)}$$

$$EV = 7.5 * pl\_actual/(pl\_planned) \quad \text{(A.4)}$$

$$AC = 0.415 * T\_f \quad \text{(A.5)}$$

末永 俊一郎 （正会員）

1974 年生．1997 東北大学理学部地球物理学科卒業．1999 同大学大学院理学研究科地球物理学専攻修士課程修了．2009 総合研究大学院大学複合科学研究科情報学専攻博士課程修了．日本ユニシス（株）等を経て，2013 年より国立情報学研究所特任准教授，現在に至る．博士（情報学）（総合研究大学院大学）．無線センサネットワーク，プロジェクトマネジメント，自己適応システムの研究に従事．

鄭 顕志

1980 年生．2003 早稲田大学理工学部情報学科卒業．2008 同大学大学院理工学研究科情報・ネットワーク専攻博士課程修了．2006 早稲田大学理工学部助手．2007 早稲田大学基幹理工学部助手．2008 早稲田大学メディアネットワークセンター助教．2010 国立情報学研究所アーキテクチャ科学研究系助教，現在に至る．博士（工学）（早稲田大学）．ソフトウェア工学，自己適応システム，無線センサネットワークの研究に従事．