大規模災害救助の分散シミュレーションカーネルの開発

Development of a Distributed Simulation Kernel for Large-scale Disaster

小藤　哲彦
Tetsuhiko KOTO

電気通信大学　情報工学科　竹内郁雄研究室
〒182-8585　東京都調布市調布ヶ丘 1-5-1
E-mail: koto @ takopen.cs.uec.ac.jp

ABSTRACT. Integrated disaster simulation systems are important tools to decrease the damage from disasters. For integrated disaster simulation systems it is highly desirable to have the following features. (1) Sub-simulators can be plugged into the system. It allows us to develop sub-simulators individually, and to build a system from suitable sub-simulators. (2) For the sake of scalable disaster simulations, it can work on a cluster of variable number of computers so that disaster over larger area can be simulated without being slowed down. I developed a kernel of an integrated simulation system with these features. The paper describes the kernel and the architecture of the simulation system. It also describes the performance of the system.

1　背景

1995 年 1 月 17 日に発生した阪神・淡路大震災は 6,000 人を超える死者、100 万人を超える被災者を出し、我が国の防災・救命システムの不備を露呈した。日本は世界有数の地震頻発国であり、過去の幾多の大地震により被った人的・経済的・精神的被害は計り知れない。さらに、宮城県沖、東海などでは近い将来にきわめて高い確率で大規模地震が発生すると予測されている。また、地震だけではなく、台風による風水害、土砂崩れ、海難事故など、災害や事故は後を絶たず、防災・救命救助システムに対する最先端技術の適用と研究はきわめて緊急性が高い。開発者は、上記の視点で大規模災害の被害を最小化する情報処理技術の発展を目指す、RoboCupRescue プロジェクト [1] に参加し、災害シミュレータの開発に携わっている。そこで作成したプロトタイプシミュレータ [2, 3] (図 1) は、サブシミュレータをプラグインできるシミュレーションシステムという第一目標は達成したものの、シミュレーションのサイズに比例して計算に時間がかかり、災害救助活動の実用に供するには能力が不足していた。

図 1: プロトタイプシミュレータのスクリーンショット (線が道路、点が家屋の代表点を表す)

2　目的

本プロジェクトの目的は、次の特徴を持つ災害シミュレーションシステムのカーネルプログラムを作成することである。

サブシミュレータのプラグイン　地震、火災、交通など、現象に応じたサブシミュレータを個別に開発し、それらを統合してシミュレーションを行うことができる。用途に応じて最適な精度・パフォーマンスを持つサブシミュレータを選択したり、新しいドメインのサブシミュレータを開発することで、世界各地で発生する様々な種類の災害をこのカーネルの上でシミュレーションしたりできる。
また、市民、消防士、救急隊員などの、知的な固体に対してサブシミュレータを用意することで、大規模なマルチエージェントシステムとして動作する。

様々な時間粒度のシミュレーションの統合　サブシミュレータの種類や、シミュレーションの状況によってシミュレーションの適切な時間粒度は異なる。粒度を一定にしてしまうと、必要以上にオーバーヘッドが大きくなる (小さな粒度で統一した場合) か、シミュレーションの精度が悪くなる (大きな粒度で統一した場合)。このため、サブシミュレータごとに、自由な時間粒度でシミュレーションを行える必要がある。

シミュレーションシステム外部からのシミュレーションへの介入　シミュレーションの最中に、シミュレーション世界の状態を変更できる。これにより、シミュレーションの

観測者がシミュレーションの状況を見て新たな設定 (例えば救援物資の到着) を導入したり、現実の災害において被害予測等を行う場合に、現実の災害の観測結果をシミュレーションにフィードバックすることができる。また、災害を擬似的に体験するシステムを構築した場合、体験者の対応をシミュレーションにフィードバックすることができる。

分散環境での動作 大規模災害における様々な現象を広範囲にシミュレーションするためには、膨大な計算が必要となる。このため、多数の計算機を利用することで、計算にかかる時間を減らせることが必要である。

このシステムによって、災害時の被害予測、救助戦略の立案支援、防災の観点からの都市計画の支援、防災訓練時における仮想的な災害の提供、分散人工知能の新しい研究フィールドの提供、などの効果が期待できる。

3 システムの概要

(1) スペースタイム

スペースタイム [4] の概念を用いてシミュレーションシステムを抽象化した。シミュレーションシステムの目的は、縦軸が対象の世界の状態を表す変数 (状態変数)、横軸が対象の世界における時刻であるような表 (スペースタイムグラフ) を埋めることである。列車のダイヤグラムは、スペースタイムグラフの一種である。

本システムでは、複数のサブシミュレータが 1 つのスペースタイムグラフを共有し、サブシミュレータ間のインタラクションは、スペースタイムグラフ上の値の読み書きによって実現される。システムを簡潔にするために、離散イベントシミュレーション [5] 等で使われるイベントの概念はサポートされない。しかし、スペースタイムグラフを用いて、イベントと同等の仕組みをユーザレベルで実現することができる。

デッドロックを避けるために、サブシミュレータはスペースタイムグラフの時刻 t の値を計算するのに、時刻 t 以降の値を使用してはならない。時刻 t の値を計算するのに、時刻 t の値を使用するサブシミュレータが複数あり、それらが互いの結果に依存している場合、デッドロックしてしまうためである。

シミュレーションの視覚化を行うプログラムや、シミュレーションの経過を記録するプログラムもサブシミュレータとして実装される。

(2) 状態変数

本システムでは、災害空間を、プロパティを持ったオブジェクトの集合としてモデル化する。建物や市民がオブジェクト、火の勢いやケガの程度がプロパティの例である。プロパティ１つ１つが、状態変数になる。サブシミュレータは、オブジェクトやプロパティの種類を自由に増やすことができる。また、オブジェクトは動的に増減させることができる。

これらのオブジェクト・プロパティは、サブシミュレータ間のインタラクションのためのものなので、サブシミュレータのシミュレーションアルゴリズムにとって、十分な記述力があるとは限らない。通常サブシミュレータは、内部的に精細なモデルを構築し、それに基づいてシミュレーションを行い、その結果をスペースタイムグラフ上の状態変数によって表現されるモデルへ変換し、スペースタイムグラフへ書き込む。

サブシミュレータのインタラクションのために、オブジェクトやプロパティの仕様について、サブシミュレータの開発者の間でコンセンサスが取れている必要がある。しかし、カーネルは単に任意の値を記述できるスペースタイムグラフを提供すればよいので、基本的にこれらの仕様について意識する必要はない。このため、今回のプロジェクトでは、カーネルの実装に必要なものを除いてオブジェクトやプロパティの仕様を決定せず、オブジェクトやプロパティに依存しない汎用的なフレームワークを提供するにとどめた。

(3) 保守的シミュレーション

本システムは、保守的なシミュレーションシステムである。すなわち、一度決定されたシミュレーション結果が、計算の進行に伴って修正されることはない。つまり、スペースタイムグラフへ値を設定すると、その値は変更できない。

値の変更を許すと、サブシミュレータはシミュレーションのやり直しをサポートしなければならなくなり、実行時コスト・開発コストが大きくなる。特に、内部的に独自のモデルを構築している場合、それらのモデルの履歴を保存しておく必要がある。

(4) 時間粒度

サブシミュレータによるスペースタイムグラフへの書き込みは、ミリ秒〜年単位の自由な時間粒度で行うことができる (さらに、将来的のバージョンでより広範囲の時間粒度を扱うように変更できる)。サブシミュレータは、スペースタイムグラフのシミュレーションに必要な範囲に値が書き込まれるまでサスペンドすることができ、他のシミュレータがより小さな粒度でシミュレーションを行っても、オーバーヘッドは大きくならない。これにより、サブシミュレータは動的に時間粒度を変更できる。

(5) 競合解消

複数のサブシミュレータが同一のプロパティに書き込みを行いたい場合がある。例えば、災害シミュレーションの場合、延焼シミュレータによって建物の火の勢いが設定されるだけでなく、着火シミュレータによっても火の勢いが設定される。このような場合に、異なるサブシミュレータからの書き込み競合を解消する仕組みを用意する必要がある。本システムでは、マージャーと呼ばれるモジュールによって、競合を解消する。マージャーはサブシミュレータの書き込みを横取りし、値をマージした結果をスペースタイムグラフへ書き込む。マージのアルゴリズムは、ユーザによりカスタマイズすることができる。

サブシミュレータにとっては、自分が書き込みを行ったプロパティの値が、自分が書き込んだ値とは異なる場合があることになる。このため、サブシミュレータは、自分が書き込んだプロパティについて、書き込みの後読み込みを行い、その値を自分のシミュレーションモデルに反映させなければならない。

このフレームワークを利用して、シミュレーションの途中経過を強制的に変更するインタラクティブ性を実現できる。変更したいシミュレーション結果をスペースタイムグラフへ書き込むプログラムをサブシミュレータとして実装し、そのサブシミュレータの書き込み結果を優先的に採用するマージャーを用意することにより、本来のシミュレーション結果を強制的に変更することができる。

また、シミュレーションの初期値を提供するモジュールをサブシミュレータとして作成することを、このサブシミュレータの書き込みを優先的に採用するマージャーを用意することで、実現した。

(6) 汎用性

実行環境がより汎用的である方が、サブシミュレータの開発にかかるコストや、シミュレーションシステムを災害

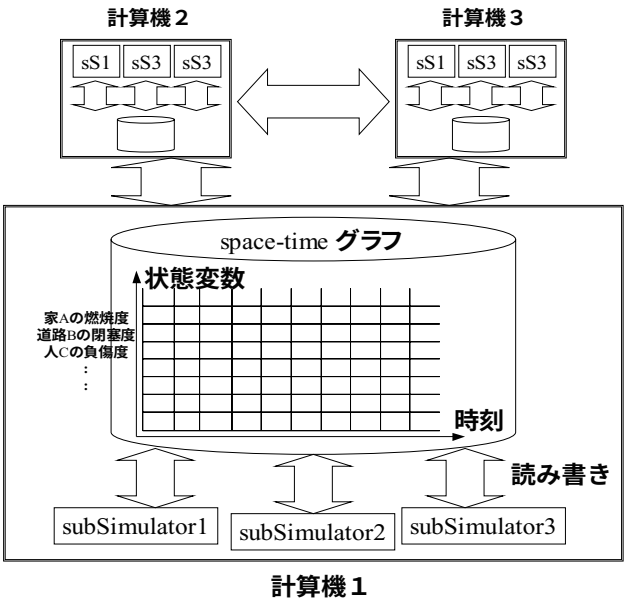

図 2: システムの概要

現場や自治体等で構築するコストが小さくなり、望ましい。Java と TCP/IP (HORB[6]) による実装を行った。

(7) **分散シミュレーション**

スペースタイムグラフは、利用可能な計算機の台数の小さなスペースタイムグラフ (サブグラフ) に分割され、各計算機に割り当てられる。各計算機は、割り当てられたサブグラフを埋め、それによって結果的にスペースタイムグラフ全体が埋まる。

サブシミュレータは、各計算機に１つずつ存在する。例えば、火災のシミュレーションを行う場合、各計算機に火災シミュレータが１つずつ存在する (図 2)。サブシミュレータは、自分が存在する計算機に割り当てられたサブグラフを埋めるために、他の計算機のサブグラフを参照する必要がある。この参照のために、計算機間で通信を行う必要があるが、シミュレーション結果を、単純にその都度通信していると、通信によるオーバーヘッドが大きくなる。本システムでは、通信を行わずに、シミュレーションによって他の計算機のサブグラフの状態を予測することで、通信オーバーヘッドを抑える。詳細を次章で説明する。

カーネルは、サブグラフの総体としてのスペースタイムグラフにアクセスするためのインターフェイスを提供しない。このため、シミュレーションの視覚化を行うサブシミュレータや、シミュレーションの経過を記録するサブシミュレータは、サブグラフから総体としてのスペースタイムグラフを独自に構築する必要がある。

4 分散シミュレーション

本シミュレーションシステムでは、スペースタイムグラフをサブグラフに分割して各計算機に割り当て、各計算機が担当するサブグラフを埋めることで、結果的にスペースタイムグラフ全体が埋まり、シミュレーションが完了する。

分割はオブジェクト単位で行われ、なるべく近いオブジェクトが同じ計算機に割り当てられるように行う。最も単純な方法は、短冊状に面積的に等分割することであり (図 3)、今回はこの分割法を使用した (モジュール化されているので、容易に変更できる)。地理的に移動するオブジェクトをサポートするために、分割は動的に変更できる。時

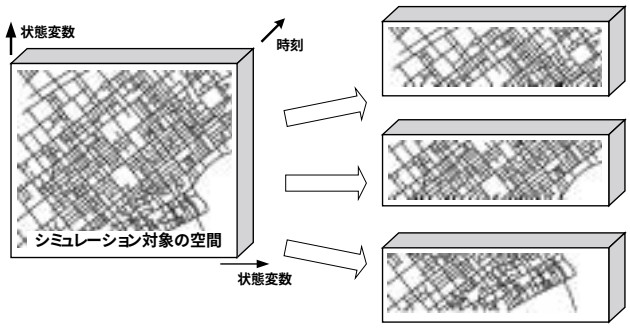

図 3: スペースタイムグラフの分割

刻を特定すると、オブジェクトは必ず１つのサブグラフに含まれ、どのサブグラフにも含まれないオブジェクトや、２つ以上のサブグラフに含まれるオブジェクトは存在しない。しかし、時刻変化に伴って、オブジェクトが異なるサブグラフへ移動しても構わない。

本システムでは、プロパティの値は、そのオブジェクトの周囲のオブジェクトから計算されることを前提とする。例えば、家の火災の進行は、その家とその家の周囲の状況のみによって決定される。この前提は、あるオブジェクトの変化が、他のオブジェクトへ影響を及ぼすのに、距離に応じた時間がかかることを意味する。ここでは、この影響の速度は高々 v であるとする。例えば、延焼シミュレーションにおける延焼速度は v を超えない。

サブシミュレータが t の時間粒度でシミュレーションを行うとき、シミュレーション対象のオブジェクトの周囲 vt の範囲にあるオブジェクトのプロパティの値を参照する必要がある。このとき、参照すべきオブジェクトが、他の計算機のサブグラフに含まれている可能性がある。

サブグラフは、各計算機に用意される別個のスペースタイムグラフによって実装される。このスペースタイムグラフには、この計算機のサブグラフに含まれるオブジェクトだけではなく、サブグラフの周囲 vt の範囲にある、シミュレーションに必要なオブジェクトのエントリも存在する (図 4)。これにより、サブシミュレータは、自分の計算機上にあるスペースタイムグラフを読み込むだけでシミュレーションが行え、読み込むオブジェクトがこの計算機が担当するサブグラフに含まれているかどうかを意識する必要がなくなる。

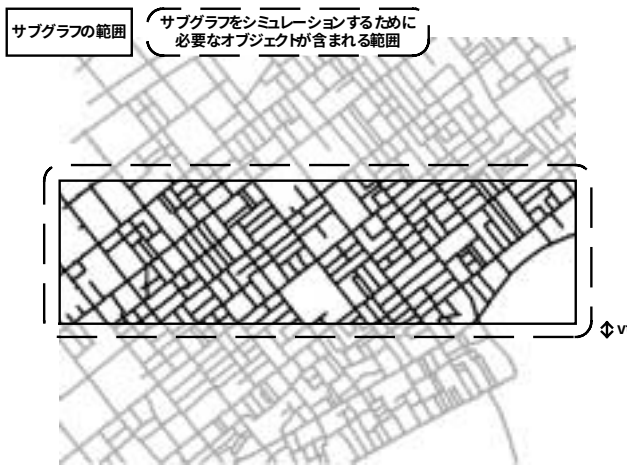

図 4: サブグラフとそのシミュレーションに必要なオブジェクトの範囲

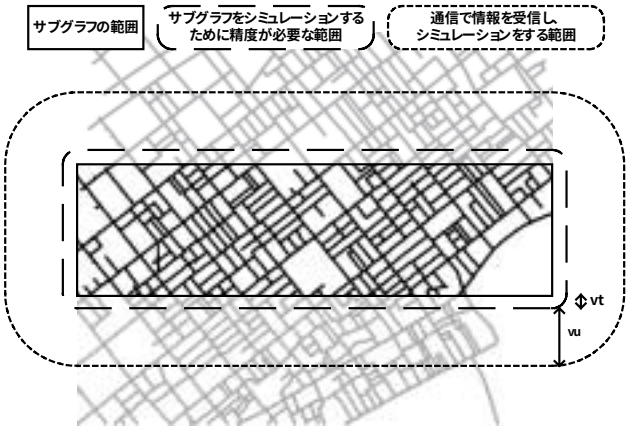

図 5: サブグラフとスペースタイムグラフの範囲

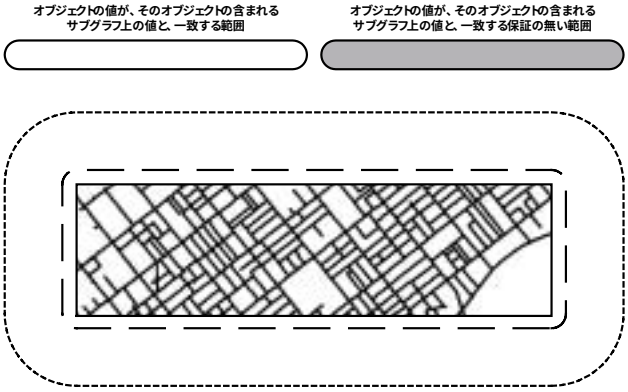

図 6: 他の計算機からサブグラフの情報を受け取った直後

この、計算機の担当するサブグラフに含まれないオブジェクトのプロパティの値は、そのオブジェクトが含まれる (他の計算機にある) サブグラフ上の値と、少なくとも「ほぼ」一致することが保証される (「ほぼ」の程度は後ほど説明する)。通常スペースタイムグラフ上で表現されるモデルは、他の計算機にある同種のサブシミュレータの内部モデルより精度が悪いので、プロパティの値が完全に一致することが保証されても、シミュレーション結果は分散を行わないときと同じにはならない。この状況では、フレームワークは、完全に一致することではなく「ほぼ」一致することのみを保証する。

しかしながら、サブシミュレータの内部モデルの情報を全てスペースタイムグラフへ書き込むようにした場合、計算機の担当するサブグラフに含まれないオブジェクトの値が、そのオブジェクトを含むサブグラフ上における値と完全に一致することが保証されれば、サブシミュレータは、分散シミュレーションを、分散せずに行ったシミュレーションと同じ結果をもたらすように行うことができる。このため、サブシミュレータの内部モデルが、スペースタイムグラフ上の値から計算可能な場合、フレームワークは、この値が完全に一致することを保証するようにする。

「ほぼ」あるいは完全に一致することは、以下で説明されるアルゴリズムによって保証される。以下、「ほぼ一致」と「完全に一致」をまとめて単に一致と言う。

サブグラフの周囲 vt の範囲にあるオブジェクトの値が、そのオブジェクトが含まれるサブグラフ上の値と一致するために、計算機間で通信を行う必要がある。スペースタイムグラフへの読み書きの度に通信を行うと、１回あたりの通信量は小さいものの通信回数が多くなり、オーバヘッドが増える可能性がある (複数のサブシミュレータが同一のプロパティに対して、ばらばらの時間粒度でアクセスを行うと、最小公倍数的に小さな粒度でのアクセスが必要になる)。

単純に送信すべきデータが一定量を超えるまで送信を行わない方法では、デッドロックが発生する可能性がある。小さな時間粒度でシミュレーションを行うサブシミュレータが計算機間で互いに依存していたとき、シミュレーションに必要になるデータが、既にシミュレーションが終了しているにもかかわらず入手できない状況に陥る。データが一定量を超えていないがために送信が行われず、データが送られてこないがためにシミュレーションを進めることができず、シミュレーションが進まないがゆえに送信すべきデータが増えず一定量を超えることがなくなる。このようなデッドロックは、タイムアウトを用いて、一致時間が経過したら、データが十分な量無くても送信することで回避

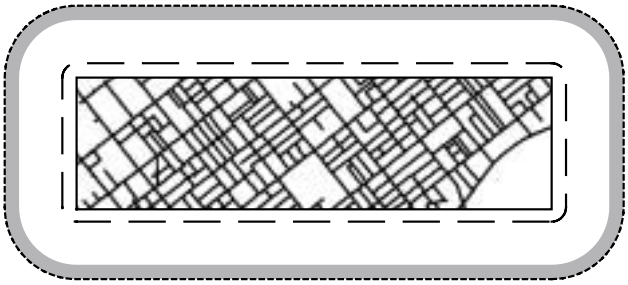

図 7: シミュレーションの実行により、周辺部分のみ予測が外れる

できる。しかしこの場合、CPU が計算を行わずに、無駄にアイドリングする可能性があり、好ましくない。

この問題を次のように解決する。通信回数を減らすために、ある程度の時間分 (シミュレーションの時間粒度より大きな値) のシミュレーションが終了するごとにしか、計算機は自分のサブグラフの値を他の計算機に通知しない。前述のデッドロックを回避するために、他の計算機から通知されたサブグラフの値を初期値としてシミュレーションを行い、通信を行わずに、他のサブグラフ上に書き込まれる値を予測する。このとき、サブグラフの周囲 vt の範囲にあるオブジェクトの値の予測が、そのオブジェクトが含まれるサブグラフ上での値に一致することを保証しなければならない。そのために、計算機は、サブグラフの値の通信において、vt よりも十分広い範囲のオブジェクトの値を送受信し、この範囲の値もシミュレーションにより予測する (図 5)。データを受信した直後 (図 6) におけるシミュレーションでは、この拡大された範囲の周辺部分のオブジェクトの値の予測は、シミュレーションに必要なオブジェクトの情報が得られないため外れる (図 7 の網掛の部分)。しかし、それよりも内側の部分のオブジェクトについては、予測は一致する。サブシミュレータの内部情報が全てスペースタイムグラフから算出可能であれば、サブシミュレータは同じ入力に対して同じアルゴリズムでシミュレーションを行うので、結果は完全に一致する。もし内部情報が算出不可能であれば、同じプロパティの値と異なる内部モデルに対して同じアルゴリズムでシミュレーションを行うので、結果は「ほぼ」一致する。「ほぼ」の程度は、このような計算が一致するような程度である。内部モデルの差異よりプロパティの値が同じであることの方が支配的に働くと予想されるので、大きく異なることはないはずである。

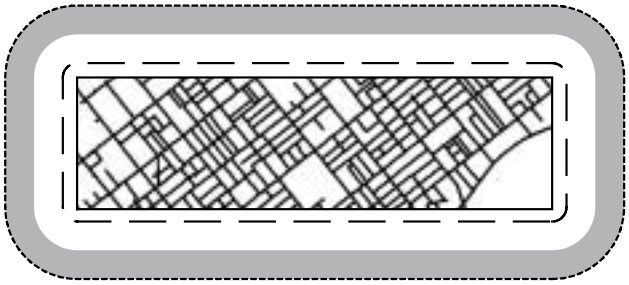

図 8: シミュレーションの進行に伴い、周辺部の予測のはずれが内側に伝播されていく

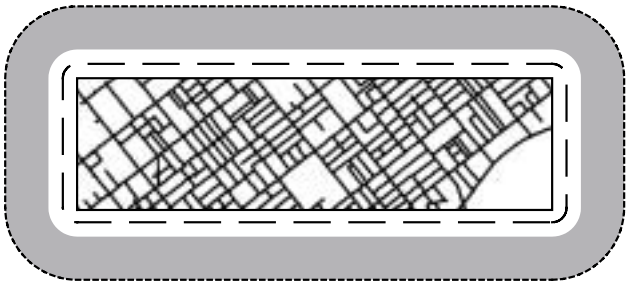

図 9: さらに内側に伝播されていく

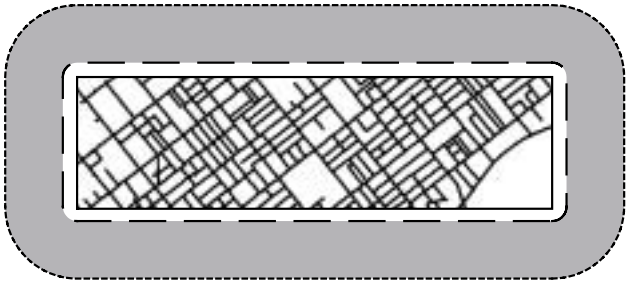

図 10: 他の計算機のサブグラフから情報を受け取らなければならない状態

シミュレーションが進行するにつれて、この予測が外れる地域は徐々に内側へ広がっていく。しかし、オブジェクトの変化が他のオブジェクトへ影響を及ぼすのに距離に応じた時間がかかることを前提にしているので、周辺部分のオブジェクトの予測失敗の影響が、vt に近い付近まで伝播するのに時間がかかる (図 8〜10)。

オブジェクトの変化が他のオブジェクトへ影響を及ぼす速度が、高々 v であるので、計算機間の通信が時間 $u(>v)$ の間隔で行われているとすると、$vu+vt$ の範囲のオブジェクトの情報を通信することで、vt の範囲にあるオブジェクトの値を、そのオブジェクトが含まれるサブグラフ上における値と一致するように保つことができる。

この方法により、通信量と計算量をふやして、通信回数を 大きく減らすことができる。

この方法のために、サブシミュレータが新たに実装すべきことは何も無い。$vu+vt$ の範囲のオブジェクトがスペースタイムグラフに用意され、サブシミュレータはスペースタイムグラフ全体に対して、サブグラフのオブジェクトであるか $vu+vt$ の範囲のオブジェクトであるかを意識せずに、シミュレーションを行い、結果をスペースタイムグラフへ書き込む。これにより、結果的に vt の範囲のオブジェクトの値は、他の計算機にあるサブグラフ上の値と一致する。

5 カーネルの実装

カーネルは３層のレイアからなっている。

(1) 最下層

最下層は、スタンドアロンのスペースタイムグラフを提供する。この層が提供するスペースタイムグラフでは、複数のサブシミュレータが同一の状態変数に書き込むことはできない。

(2) 中間層

中間層は、最下層のスペースタイムグラフを使用して、異なるサブシミュレータが同一の状態変数に書き込むことのできる、スタンドアロンのスペースタイムグラフを提供する。複数のサブシミュレータが同じ状態変数に対して書き込み権を取得した場合に、それぞれを、最下層が提供するスペースタイムグラフの異なる状態変数にマッピングする。そして、それらの状態変数に対する書き込みをマージするサブシミュレータ (最下層が提供するスペースタイムグラフに対するサブシミュレータ) を用意する。このサブシミュレータが、マージャーである。マージのアルゴリズムには、最大値や平均を取るもの、優先順位によってどれか１つのサブシミュレータの結果のみを採用するものなどが考えられ、ユーザによってカスタマイズ可能である。マージャーは、例外的に時刻 t の値を計算するのに時刻 t の値を使用する。

(3) 最上層

最上層では、中間層の提供するスペースタイムグラフを使用して、分散環境で動作するスペースタイムグラフを提供する。各サブグラフの境界から一定の範囲にあるオブジェクトの情報を、定期的に送信する。この層は、シミュレーションシステム全体のスペースタイムグラフを、どうサブグラフに分割するかも決定する。

(4) インターフェイス

以下は、スペースタイムグラフの基本的なインターフェイスの Java に基づいた記述である。全ての層でこのインターフェイスは共通である。

```
// A Interface for the Initialization
interface Memory {
    Writable registerWriter(
            String propertyName,
            MemoryWriter writer);
    void registerReader(
            String propertyName,
            CalledBackMemoryReader reader);
    Readable registerReader(
            String propertyName,
            MemoryReader reader);
}

// 3 Interfaces for the Simulation Progress
interface Writable {
    void writeToMemory(...);
}
interface CalledBackMemoryReader {
    void written(...);
}
interface Readable {
    Object getValue(String objectID, Time time);
    Object getAvarageValue(
            String objectID,
            Time start,
            Time end);
```

```
    void discardBefore(Time time);
    UpperBoundary waitFixed(Time time);
}
```

Memory は、スペースタイムグラフを表すインターフェイスであるが、Memory 自体はシステムの初期化時にしかアクセスされない。サブシミュレータは初期化時に、registerWriter, registerReader メソッドによって、Memory に自分自身を登録し、スペースタイムグラフへのアクセス権を得る。メソッドの引数には、アクセスするプロパティの名前を指定する。複数のプロパティにアクセスする場合、複数回メソッドを呼び出す必要がある。

registerWriter メソッドによって書き込みを登録すると、Writable が返される。サブシミュレータは、Writable のメソッドを呼び出すことによって、スペースタイムグラフに対する書き込みを行うことができる。

registerReader メソッドは 2 種類ある。1 つめのメソッドによって読み込みを登録すると、登録したプロパティに対して書き込みがある毎に、registerReader メソッドに渡した CalledBackMemoryReader オブジェクトに対して、メソッド呼び出しが行われる。メモリ変化の差分のみを受け取ることになり、サブシミュレータを作りにくい。例えば、あるオブジェクトの周囲のオブジェクトについて知りたい場合、差分を網羅的に検索するか、あらかじめ別のデータ構造に変換しておく必要がある。

2 つめの registerReader メソッドは、Readable を返す。サブシミュレータは、Readable のメソッドを呼び出すことによって、スペースタイムグラフを読むことができる。また、discardBefore メソッドによって、もう読み込みを行わないスペースタイムグラフの領域を宣言することができ、その領域の値を記憶するために使われていた計算機のメモリを開放することができる。また、waitFixed メソッドによって、読み込みたい領域が他のサブシミュレータによって書き込まれるまで、サブシミュレータをサスペンドすることができる。2 つめの registerReader は、1 つめの registerReader とは異なり、書き込みをメモリ上に保持するので、メモリ使用量が多い。また、若干のオーバーヘッドがある。

シミュレーション世界中のオブジェクトの増減は、org.robocup.rescue.System オブジェクトの org.robocup.rescue.addedObjects プロパティと org.robocup.rescue.removedObjectIDs プロパティに対する書き込みによって実現される。シミュレーション対象の世界にどのようなオブジェクトが存在しているかは、org.robocup.rescue.objectIDs プロパティに記録される。

6 実験

(1) 実験の内容

台数効果を測定するために、以下の実験を行った。

実験環境として、1 つの Pentium III 900MHz、256MB のメモリ、Linux 2.2.17 を搭載した PC を 8 台用意し、1 台のスイッチングハブを用いて、100BASE-TX のネットワークを構成した。

アプリケーションプログラムとして、簡単な災害シミュレーションを行うサブシミュレータを作成し接続した。サブシミュレータは、火災の延焼をシミュレーションするものと、市民の移動をシミュレーションするものと、シミュレーションの初期値を設定するサブシミュレータの 3 つを用意し、初期値として、阪神・淡路大震災が発生する前の神戸市長田区の地図 (1.5km×1.5km) を用意した。

1〜8 台の PC を用いて、

$$\frac{1.5\times(\text{PC の台数})}{8}\text{km} \times 1.5\text{km}$$

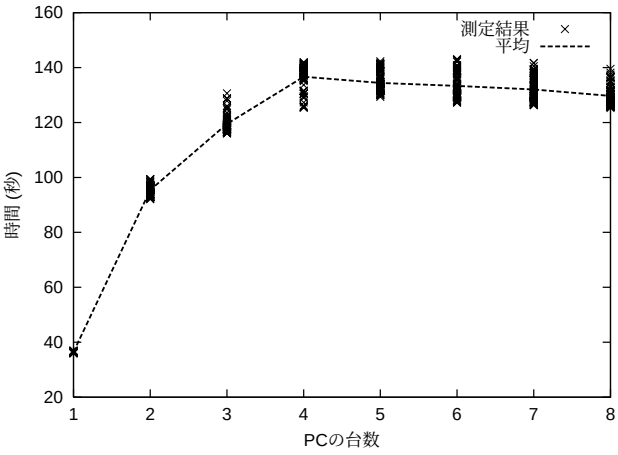

図 11: PC の台数に対する計算時間

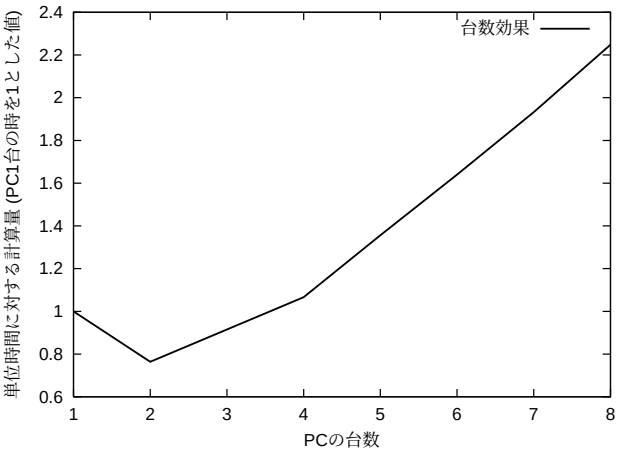

図 12: 台数効果

の範囲のシミュレーションを行った。領域を短冊状に区切り、各 PC では 1.5/8km×1.5km の範囲をサブグラフとしてシミュレーションを行った。1 台の PC で 1.5km×1.5km のシミュレーションを行うと、メモリのスワップにより正当な評価が難しくなるため、台数に応じてシミュレーション範囲を広げるようにした。

30 ステップのシミュレーションを、80 回ずつ行い、シミュレーションが終了するまでの時間を測定した。

(2) 結果と考察

図 11 の結果が得られた。また、これをもとに台数効果を計算したものが図 12 である。

図 11 では、台数に応じてシミュレーションの量をふやしているため、グラフが右肩上がりであれば台数効果が低く、水平であれば台数に比例したパフォーマンスを発揮していることになる。PC4 台まではグラフが右肩上がりになっており、台数効果は低い。しかし 4 台以降ではグラフはほぼ水平になっており、この範囲では、n 台の PC によるシミュレーションと m 台の PC によるシミュレーションを比べると、単位時間あたりの計算量はほぼ m/n 倍になる。シミュレーションの規模に対して計算機の台数を増やすことで、計算時間の増加を抑えることができていると言える。

しかし、PC1 台のみでシミュレーションを行った場合と比較すると、8 台に対して約 2.5 倍と、台数効果は低くなっている。これは、PC1 台のみでシミュレーションを行う場合、ネットワークを用いた通信や、サブグラフ周囲の

シミュレーションを行う必要が無いことが大きく影響していると考えられる。

7 関連研究

(1) HLA

複数のシミュレータを結合してシミュレーションを行うシステムに、High Level Architecture[7, 8, 9] (HLA) がある。HLA は、本プロジェクトのシステムと同様、サブシミュレータ (federate と呼ばれる) をプラグインすることができる。しかし、HLA はスペースタイムグラフを提供せず、値の履歴を元にしたシミュレーションをサポートしない。このため、値の履歴を視野に入れたより高精度のシミュレーションや、変動の緩い付近を大きな時間粒度でシミュレーションする等のパフォーマンス向上を行う余地が無い。

(2) FUSS

同様に、複数のシミュレータを結合してシミュレーションを行うシステムとして、FUSS[10] が提案されているが、大規模なシミュレーション対象を想定しておらず、スケーラビリティに欠ける。

8 将来への課題

(1) 負荷の均等化

シミュレーション対象の空間を、単純に面積によってサブグラフへ分割しても、負荷が適切に分割されるとは限らない。災害シミュレーションの場合、火災の発生している付近や、人の多い付近のシミュレーションにより多くの計算が必要となり、その位置は動的に変化する。このため、動的にサブグラフの境界線を変更し、計算機の負荷が均等になるようなアルゴリズムを導入することが望ましい。現在フレームワークとしては、動的な境界線の変更をサポートしているが、その変更をどのようなアルゴリズムによって決定するのが良いかは、自明ではない。

(2) 動的拡張性

本システムでは、システムの初期化時にプラグインするサブシミュレータが決定され、シミュレーション開始後にサブシミュレータを抜き差しすることはできない。現実の災害において、対策本部における意思決定支援等に使用されることを想定すると、シミュレーションの最中に、それを見ている人間によってシミュレーションに新たな設定を導入できるインタラクティブ性が求められる。そのためには、サブシミュレータを動的にプラグインできるアーキテクチャが必要となる。

(3) ロールバック機能

同様に、インタラクティブ性のために、特定の時刻までシミュレーションをロールバックする機能があることが必要である。特定時刻までのスペースタイムグラフを初期値としてシミュレーションを開始することで、擬似的なロールバックを行えるが、十分な精度が得られるか調査が必要である。

(4) マルチエージェントシステムとしての洗練

シミュレーション対象の世界の中の、市民や消防士などのエージェントのシミュレーションについて、積極的なサポートを行わなかった。しかし、エージェントを扱うサブシミュレータには、多くの共通の問題があり、それらをフレームワークとしてサポートすることが望ましい。

災害シミュレーションにおいては、エージェント間の通信は、重要なシミュレーション要素である。しかし、本プロジェクトで開発したシステムは、地理的に離れた位置にあるオブジェクト間に直接の相互作用がない、あるいは相互作用の伝達に時間がかかることを前提にして成立している。エージェント間の通信は、地理的に離れているにもかかわらず、短時間で行われるため、特別な拡張が必要になる (通信回数・通信量が問題になるが、実時間性を利用することで通信を抑えるアプローチが考えられる)。また、エージェント間の無線などを用いた通信は、メディア (空気、電線、電磁場等)、言語 (日本語、英語、方言等)、知識などによって成否がきまり、これらの概念を高度に抽象化した、通信の成否をシミュレーションするためのフレームワークが作成できる可能性がある。

また、エージェントが地理的に移動した場合、ある地域のシミュレーションを行っているプロセスと、その地域に存在するエージェントをコントロールするプロセスが、異なる計算機上に位置することになる。この場合に、モバイルエージェント等の技術によってプロセスを移動することによって、パフォーマンスが向上する可能性がある。

(5) 型付のモデル

現在のシステムではオブジェクトやプロパティに型がなく、オブジェクトやプロパティを型に縛られずに自由に作成することができる。サブシミュレータを開発する際に、間違って、ありえないオブジェクトやプロパティを作ってしまうと、しばしばそのバグを発見することが困難である。このため、オブジェクトやプロパティに型をもたせ、静的あるいは動的にチェックできることが望ましい。

(6) ユーティリティメソッド・クラスの追加

現在のシステムは必要最低限のプリミティブしか提供していない。このため、実際にサブシミュレータを開発するのは、大変な作業であり、開発を容易にするユーティリティの追加が必要である。具体的には、シミュレーション対象の世界のオブジェクトを、Java のオブジェクトとしてアクセスできる機能などが考えられる。

(7) 他言語のサポート

現在のシステムは Java のみをサポートしている。既に他の言語で開発されているシミュレータの接続を可能にするために、様々な言語をサポートすることが望ましい。

9 今後の展開

RoboCupRescue プロジェクトの一端として、社会に利益をもたらすことができるように活動を行っていく。

前述の課題の克服とともに、各種のサブシミュレータを開発し、災害シミュレーションシステムを構築し、防災訓練のための仮想的な災害の提供などの啓蒙活動や、防災の観点からの都市計画の支援などを目指す。さらに将来的には、災害が発生した際に、災害の情報を取り込みながらシミュレーションを行い、災害対応を支援するシステムを目指す。

RoboCupRescue では、世界各国の参加者により作成された、レスキュー活動を行うエージェントプログラムの優劣を競う競技会を開催しており、本プロジェクトで製作したシミュレーションカーネルは、2002 年以降の大会において使用される予定である。

10 参加企業及び機関

なし。

11. 参考文献

[1] 田所諭, 北野宏明他. ロボカップレスキュー. 共立出版, 2000.

[2] M. Ohta, T. Koto, I. Takeuchi, T. Takahashi and H. Kitano. Design and Implementation of the Kernel and Agents for the RoboCup-Rescue. In Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000), 2000.

[3] 小藤哲彦, 竹内郁雄, 北野宏明. カーネルの通信と役割. 情報処理学会第 60 回全国大会, 2000.

[4] K. M. Chandy and R. Sherman. Space, time, and simulation. Proceedings of the SCS Multiconference on Distributed Simulation, Vol. 21, No. 2, pp. 53–57, March 1989.

[5] Richard M. Fujimoto. Parallel discrete event simulation. Communications of ACM, Vol. 33, No. 10, pp. 30–53, October 1990.

[6] http://www.horb.org/.

[7] IEEE Std 1516-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.

[8] IEEE Std 1516.1-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification.

[9] IEEE Std 1516.2-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification.

[10] NODA, Itsuki. Framework of Distributed Simulation System for Multi-agent Environment. In RoboCup 2000: Robot Soccer World Cup IV. Springer, 2001.