(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-521411
(P2004-521411A)
(43)公表日　平成16年7月15日(2004.7.15)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 9/44	G06F 9/06	620A	
G06F 15/16	G06F 9/44	530P	
	G06F 9/44	535	
	G06F 15/16	620T	
	G06F 9/06	620K	

審査請求 有　予備審査請求 有　（全 58 頁）　最終頁に続く

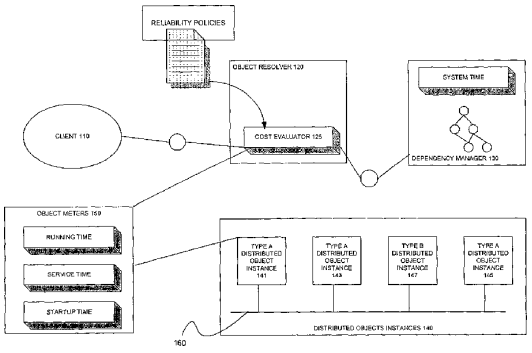
(21)出願番号　特願2002-553637(P2002-553637)
(86)(22)出願日　平成13年11月13日(2001.11.13)
(85)翻訳文提出日　平成15年6月20日(2003.6.20)
(86)国際出願番号　PCT/US2001/043640
(87)国際公開番号　WO2002/052403
(87)国際公開日　平成14年7月4日(2002.7.4)
(31)優先権主張番号　09/741,869
(32)優先日　平成12年12月22日(2000.12.22)
(33)優先権主張国　米国(US)

(71)出願人　503161132
インテル・コーポレーション
アメリカ合衆国・カリフォルニア・９５０５４・サンタ・クララ・ミッション・カレッジ・ブールヴァード・２２００
(74)代理人　100064908
弁理士　志賀　正武
(74)代理人　100108578
弁理士　高橋　詔男
(74)代理人　100089037
弁理士　渡邊　隆
(74)代理人　100101465
弁理士　青山　正和
(74)代理人　100094400
弁理士　鈴木　三義

最終頁に続く

(54)【発明の名称】分散プログラミングネットワークにおける適応信頼性バランシングのためのシステム及び方法

(57)【要約】
本願発明の代表的な実施形態は、過去の分散プログラミングネットワークコンポーネントヒストリーに基づき、信頼性バランシングを実行するための方法およびシステムを提供する。その過去の分散プログラミングネットワークコンポーネントヒストリーは、これらの資源の有用性および信頼性を改善する目的で、コンピューティング資源およびそれらのプロセシングコンポーネントをバランスさせる。即ち、サービスの要求を受けるステップと、要求されたサービスに関連するオブジェクトインスタンスを確認するステップと、オブジェクトインスタンスとサービスとの間の依存性を確認するデータについて質問するステップと、確認されたオブジェクトインスタンスに関連する信頼性メトリックについて質問するステップと、信頼性メトリックに基づき何れのオブジェクトインスタンスがサービスを実行し得るかを決定するステップとを備える。



【特許請求の範囲】
【請求項１】
分散プログラミングネットワークにおいて信頼性バランシングを実施するための方法であって、
サービスの要求を受けるステップと、
前記要求されたサービスに関連する少なくともひとつのオブジェクトインスタンスを確認するステップと、
前記少なくともひとつのオブジェクトインスタンスと前記要求されたサービスとの間の依存性を確認するデータについて質問するステップと、
前記確認された少なくともひとつのオブジェクトインスタンスに関連する少なくともひとつの信頼性メトリックについて質問するステップと、
少なくともひとつの信頼性メトリックに基づき何れのオブジェクトインスタンスが最も確かに前記サービスを実行し得るかを決定するステップと
を備えた方法。
【請求項２】
何れのオブジェクトインスタンスがより確かに前記サービスを実行し得るかを決定することは、また、前記少なくともひとつのオブジェクトインスタンスと前記要求されたサービスとの間の依存性に基づくことを特徴とする請求項１に記載された方法。
【請求項３】
何れのオブジェクトインスタンスがより確かに前記サービスを実行し得るかを決定することは、また、前記分散プログラミングネットワークの信頼性ポリシーに基づくことを特徴とする請求項１に記載された方法。
【請求項４】
前記少なくともひとつの信頼性メトリックに基づき何れのオブジェクトインスタンスが最も確かに前記サービスを実行し得るかの前記決定に基づき、前記サービスの要求を少なくともひとつのオブジェクトインスタンスとマッチングさせることを更に具備することを特徴とする請求項１に記載された方法。
【請求項５】
前記サービスの要求をマッチングさせるステップは、少なくともひとつのヒストリー、および前記分散プログラミングネットワークに含まれるオブジェクトインスタンスに関する将来のサービス要求の統計学上の予測に対応する少なくともひとつの信頼性メトリックを評価することを含むことを特徴とする請求項４に記載された方法。
【請求項６】
オペレーティング分散プログラミングネットワークにおける信頼性バランシングを実施するために構成されたシステムであって、
コントロールファブリックを介して互いに結合された複数のオブジェクトインスタンスから要求されたサービスに関連する少なくともひとつのオブジェクトインスタンスを確認すると共に、前記確認された少なくともひとつのオブジェクトインスタンスに関連する少なくともひとつの信頼性メトリックについて質問し、且つ何れのオブジェクトが前記サービスの要求を最も確かに実行し得るかについて決定するように構成されたオブジェクトレゾルバと、
前記オブジェクトレゾルバと結合され、前記少なくともひとつのオブジェクトインスタンスと前記要求されたサービスとの間の依存性を確認するデータを提供するように構成された依存性マネージャと、
少なくともひとつのオブジェクトインスタンスに関する前記少なくともひとつの信頼性メトリックを発生する少なくともひとつのオブジェクトメータと
を具備するシステム。
【請求項７】
前記オブジェクトレゾルバは、前記分散プログラミングネットワーク特有の信頼性ポリシーへのアクセス権を有するコストエバリュエータを含むことを特徴とする請求項６に記載

されたシステム。
【請求項８】
前記分散プログラミングネットワーク内のオブジェクトおよびオブジェクトインスタンスに対応する累積信頼メトリックを供給するために有効性メトリックアクロスパワー(across power)およびシステム故障を保持するように構成されたことを特徴とする請求項６に記載されたシステム。
【請求項９】
動的信頼性バランシングを提供するために前記分散プログラミングネットワークの継続的モニタリングを実施することを特徴とする請求項６に記載されたシステム。
【請求項１０】
サービスの要求とオブジェクトとの間のマッチングを実施して、前記要求されたサービスを供給するための少なくともひとつのオブジェクトインスタンスの有用性を評価することにより前記サービスの要求を実行することを特徴とする請求項６に記載されたシステム。
【請求項１１】
前記オブジェクトインスタンスの有用性は、故障までの平均時間および修復までの平均時間に基づき算出されることを特徴とする請求項１０に記載されたシステム。
【請求項１２】
前記オブジェクトインスタンスの有用性は、故障までの平均時間と修復までの平均時間との和により、前記故障までの平均時間を除算したものとして算出されることを特徴とする請求項１０に記載されたシステム。
【請求項１３】
前記故障までの平均時間は、初期の瞬間から次の故障イベントまでの時間間隔であることを特徴とする請求項１２に記載されたシステム。
【請求項１４】
前記故障までの平均時間は、システムサービス信頼性を統計学的に数量化したものであることを特徴とする請求項１３に記載されたシステム。
【請求項１５】
前記故障までの平均時間は、故障から回復し且つサービスの実施を回復させるための時間であることを特徴とする請求項１２に記載されたシステム。
【請求項１６】
サービスの実施は、前記要求されたサービスを供給するために協調して動作するオブジェクトが特定の前記要求されたサービスを指定されたように提供したときに達成されることを特徴とする請求項１５に記載されたシステム。
【請求項１７】
前記修復までの平均時間は、サービスの中断を統計学的に数量化したものであることを特徴とする請求項１２に記載されたシステム。
【請求項１８】
前記オブジェクトレゾルバは、少なくともひとつのオブジェクトインスタンスまたはオブジェクトインスタンスのグループの動作に関するリアルタイムデータを評価することを特徴とする請求項６に記載されたシステム。
【請求項１９】
前記分散プログラミングネットワークの特性を変えることに基づきサービス要求ルーティングの適応を可能とすることを特徴とする請求項１８に記載されたシステム。
【請求項２０】
適応はリアルタイムに実施されることを特徴とする請求項１９に記載されたシステム。
【請求項２１】
前記サービスの要求は、前記分散されたオブジェクトの１または２以上の中から要求された使用するものを捜し又は有する、アプリケーションまたは分散プログラミングネットワークオブジェクトから生じることを特徴とする請求項６に記載されたシステム。
【請求項２２】

サービスの要求はクライアントから生じ、前記クライアントは、そのクライアントによって期待される信頼性のレベルを示す少なくともひとつの信頼性の制約を発生し又は割り当てられることを特徴とする請求項６に記載されたシステム。
【請求項２３】
前記オブジェクトレゾルバは、前記クライアントにより供給される少なくともひとつの信頼性の制約を満たす、特定オブジェクトおよびそのオブジェクトのインスタンスを示すレファレンス確認データを返すサービスであることを特徴とする請求項６に記載されたシステム。
【請求項２４】
前記オブジェクトレゾルバは、リファレンス確認データを返すサービスであり、前記リファレンス確認データは、前記サービスの要求において供給される少なくともひとつの信頼性の制約を満たす、特定オブジェクトおよびそのオブジェクトのインスタンスを含むことを特徴とする請求項６に記載されたシステム。
【請求項２５】
前記依存性マネージャは、前記分散プログラミングネットワークに含まれる分散オブジェクト間の、トポロジーおよび依存性に関して受容性のあるサービスであることを特徴とする請求項６に記載されたシステム。
【請求項２６】
前記オブジェクトレゾルバは、全分散プログラミングネットワークの要求を満たす最適なオブジェクトへの問い合わせを発生することを特徴とする請求項６に記載されたシステム。
【請求項２７】
前記全分散プログラミングネットワークの要件は、少なくともひとつの信頼性ポリシーを有することを特徴とする請求項２６に記載されたシステム。
【請求項２８】
前記データ確認依存性は、オブジェクトインスタンスが依存していることの一覧を有することを特徴とする請求項６に記載されたシステム。
【請求項２９】
前記少なくともひとつのオブジェクトメーターは、時間について累積的な少なくともひとつの信頼性メトリックを発生することを特徴とする請求項６に記載されたシステム。
【請求項３０】
前記少なくともひとつの信頼性メトリックは、サービス滞在時間を含みまたは該サービス滞在時間に基づくことを特徴とする請求項６に記載されたシステム。
【請求項３１】
前記少なくともひとつの信頼性メトリックは、サービス実施時間を含みまたは該サービス実施時間に基づくことを特徴とする請求項６に記載されたシステム。
【請求項３２】
前記少なくともひとつの信頼性メトリックは、開始時間を含みまたは該開始時間に基づくことを特徴とする請求項６に記載されたシステム。
【請求項３３】
分散プログラミングネットワークにおける欠陥耐性を改善する欠陥耐性サブシステムであって、
少なくともひとつのオブジェクトインスタンスと要求されたサービスとの間の依存性を確認するデータを供給するために構成された依存性マネージャを含む分散プログラミングネットワーク内でオブジェクトグループマネージメントを実行するように構成されたレプリケーションマネージャと、
前記レプリケーションマネージャからの質問を受け且つ該質問に応答し、前記少なくともひとつのフォールトディテクタの管理下で前記分散プログラミングネットワーク内のオブジェクトおよびオブジェクトインスタンスのステータスをモニタするように構成されると共に、前記分散プログラミングネットワーク内の少なくともひとつのオブジェクトに関す

る前記少なくともひとつの信頼性メトリックを発生するように構成された少なくともひとつのフォールトディテクタと、
前記レプリケーションマネージャおよび少なくともひとつのフォールトディテクタに接続され、かつ前記少なくともひとつのフォールトディテクタからのこのような欠陥の検出を示すデータを受け取ることに続いて、オブジェクトまたはオブジェクトインスタンス故障の前記レプリケーションマネージャに通知することにより前記少なくともひとつの欠陥ディテクタのための故障通知ハブとして動作するように構成されたフォールト通知部と、
複数のオブジェクトインスタンスから要求されたサービスに関連する少なくともひとつのオブジェクトを確認すると共に、前記確認された少なくともひとつのオブジェクトインスタンスに関する少なくともひとつの信頼性メトリックを質問し、かつ何れのオブジェクトインスタンスが最も確かに前記サービスの要求を実施するかについて決定するように構成された適応プレイサーと、
を備えた欠陥耐性サブシステム。
【請求項３４】
前記オブジェクトレゾルバは、前記分散プログラミングネットワーク特有の信頼性ポリシーにアクセスする権利を有するコストエバリュエータを備えたことを特徴とする請求項３３に記載された欠陥耐性サブシステム。
【請求項３５】
前記サービスの要求は、クライアントから生じ、そのクライアントは、そのクライアントにより期待される信頼性のレベルを示す少なくともひとつの信頼性の制約を発生し又は割り当てられることを特徴とする請求項３３に記載された欠陥耐性サブシステム。
【請求項３６】
前記オブジェクトレゾルバは、前記クライアントにより供給された前記少なくともひとつの信頼性の制約を満たす、特定オブジェクトおよびそのオブジェクトのインスタンスを含むリファレンス確認データを返すサービスであることを特徴とする請求項３５に記載された欠陥耐性サブシステム。
【請求項３７】
前記依存性マネージャは、前記分散プログラミングネットワークに含まれる分散オブジェクトインスタンス間の前記トポロジーおよび依存性に関して受容性を有するサービスであることを特徴とする請求項３５に記載された欠陥耐性サブシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、分散プログラミングネットワークにおける信頼性バランシングに関する。更に詳しくは、本発明は、過去の分散プログラミングネットワーク及び／又は分散プログラミングネットワークコンポーネントヒストリーに基づく分散プログラミングネットワークにおける信頼性バランシングに関する。
【背景技術】
【０００２】
デスクトップ上の低コストのコンピュータパワーよりも先にコンピューティングすることがセンタのロジカルなエリアにおいて組織化された。これらのセンタがまだ存在しているにもかかわらず、大小の企業は、彼らが企業において最も効率良く操作できる場所や、デスクトップワークステーション、ローカルエリアネットワークサーバ、地域サーバ、ウェブサーバおよび他のサーバのいくつかを複合した所に、多くの時間をかけてアプリケーション及びデータを分散させている。分散プログラミングネットワークモデルにおいて、コンピューティングは、コンピュータで処理中のコンピュータプログラミングおよびデータが１台よりも多くのコンピュータに、通常、ネットワーク上に分配された場合に、“分散された(distributed)”と言われる。
【０００３】
クライアントサーバコンピューティングは、単純に、クライアントマシンまたはアプリケ

ーションがユーザに或る能力を提供し、そしてクライアントマシンやアプリケーションにサービスを提供する他のマシンやアプリケーションから他の事柄を要求するという考えである。
今日、主要なソフトウェアメーカは、分散コンピューティングのオブジェクト指向の展望を推し進めている。Ｊａｖａ（登録商標）を用いた分散パブリッシング環境や、企業が分散アプリケーションを作り出すことを手助けする他の製品のように、ＷＷＷ(World Wide Web)が、分散コンピューティングに向けてトレンドを加速させている。分散ソフトウェアモデルもまた、大容量または使命重大な(mission critical)システムのための、拡張性があって高度に有用なシステムを提供するのに向いている。

【０００４】

ＣＯＲＢＡ(Common Object Request Architecture)は、ネットワークにおける分散プログラムオブジェクトを作成し、分散し、そして管理するためのアーキテクチャであり、仕様規格である。これにより、異なる場所で異なるベンダーにより開発された複数のプログラムが、“インターフェイスブローカ(interface broker)”を通じてネットワークにおいて通信することを可能とする。ＩＳＯ(International Organization for Standardization)は、ＣＯＲＢＡを分散オブジェクト（これはネットワークコンポーネントとしても知られている）として公認している。

【０００５】

ＣＯＲＢＡにおける本質的な概念は、ＯＲＢ(Object Request Broker)である。異なるコンピュータ上でクライアントおよびサーバのネットワークにおけるＯＲＢの支援は、クライアントプログラム（それ自体オブジェクトであってもよい）が、その物理的ロケーションまたはその装置に関わりなく、サーバプログラムまたはオブジェクトからサービスを要求できることを意味する。ＣＯＲＢＡにおいて、ＯＲＢは、分散オブジェクトまたはコンポーネントからのクライアントのサービス（例えば、まとまりのあるソフトウェア機能の集まり、その機能は共に複数のクライアントに対しサーバのような能力を呈する；サービスは、例えばそのクライアントによって遠く離れて実施可能であってもよい）に対する要求と要求達成との間で“ブローカ(broker)”として動作する。このように、ネットワークコンポーネントは、稼動しているときにお互いを見つけてインタフェース情報を交換することができる。ＯＲＢ間で要求または応答を行うため、ＧＩＯＰ(General Inter-ORB Protocol)、およびインターネットにはそのＩＩＯＰ(Internet Inter-ORB Protocol)が使用される。ＩＩＯＰは、ＧＩＯＰの要求および応答を、各コンピュータにおけるインターネットのＴＣＰ(Transmission Control Protocol)レイヤに移す(map)。

【０００６】

どのようなフレームワークまたはアーキテクチャが分散プログラミングで使用されているかに関係なく、オブジェクト指向のプログラミングにおける最初のステップは、操作すべきシステムで利用されている全てのオブジェクトを確認し、そしてそれらが互いにどのように関連づけられているかを確認することであり、これは大抵の場合データモデリングとして知られている実践である。一旦オブジェクトが確認されると、このオブジェクトの確認は、オブジェクトのクラスとして一般化され、それが含むデータのタイプおよびそのデータを操作できるいくつかのロジックシーケンスが定義される。クラスの真のインスタンス(real instance of class)は、“オブジェクト”、又はある環境においては“クラスのインスタンス”と呼ばれる。負荷バランシングおよび信頼性バランシング（この明細書で説明される）、同一オブジェクトの複数インスタンスが、分散プログラミングネットワーク内の種々の点で動作する。

【０００７】

大規模分散プログラミングシステムを管理するための二つの主要な挑戦がある。ひとつは、分散プログラミングネットワークサービスに対する要求が高い場合に性能をハイレベルに維持することである。この挑戦は、しばしば“負荷バランシング”として引用され、そして、通常よりも多くのクライアントの要求に対し、有限の分散プログラミングネットワーク資源（分散プログラミングネットワークサービスに関連した資源）の配分をバランス

させることを要求する。多くの場合、大規模な分散プログラミングネットワークは、そのサービスを受けるクライアントにサービスを提供する。この負荷要求の統計学上のバランシングは、一般に観測されており、良く研究された現象である。

【０００８】

他方の主要な挑戦は、これらの大規模分散プログラミングネットワークの継続的な動作を維持することである。この挑戦は、“信頼性バランシング(reliability balancing)”として引用される。大規模システムが、欠陥、即ちサービスエラーの原因をより抱えやすくなることはよく知られた重要事項である。加えて、より大きなシステムでは、欠陥は、そのサービスの顧客に対してより重大な影響を与えやすい。例えば、もしサービスが１つよりも多くのオブジェクトを利用またはアクセスする資源を必要とすれば、それら複数のオブジェクト中の１つに存在する故障がシステムの故障をもたらし得る。

【０００９】

従来、大規模分散プログラミングネットワークの負荷バランスの問題を解決するための多くのアプローチがある。そのどれもが完璧ではないように見える一方、それらは、その利益を論証するには有効である。しかしながら、分散プログラミングネットワークの信頼性を提供することの挑戦、即ち分散プログラミングネットワークマネージメントの働きを維持することは、決して速くはないにしても発達し切っている。大規模分散プログラミングネットワーク信頼性を提供するための従来方法および分散プログラミングネットワークが変化している。多くの主要な技術が存在する。

【００１０】

大規模分散プログラミングネットワーク信頼性を提供するための最も一般的な技術は、エンティティー(entity)、またはオブジェクトインスタンスのリダンダンシー(object instance redundancy)に依存している。この技術は、またしばしば“レプリケーション(replication)”として引用され、そして、オブジェクトまたはオブジェクトのグループの主要なインスタンスがフェイルしたときに１又は２以上の代わりのインスタンスが主要なインスタンスが切り離されたサービスを再開できることを期待して、同一のオブジェクトまたはオブジェクトのグループに代わるインスタンスを提供することにより、大きく重大なシステムにおけるコンポーネントの故障に対する或る程度の防御を提供する。

【００１１】

Ｎ－バージョンプログラミングと呼ばれるもう一つの共有技術は、同時的に実施する同一サービス（またはオブジェクト）の３つまたはそれ以上の異なるバージョン（装置）を頼みとするものである。これらの動作は、並列装置のそれぞれが、例えば他方対して一方が先行することのない同一のシーケンシングを通じて論理的に稼動するように、或るロックステップ制御(rock-step control)メカニズムを通じて制御される。適切な時間上の点で、３つまたはそれ以上のインスタンスの各出力が投じられる。その期待するところは、３つの全てのインスタンスが、それらが提供する如何なる計算作業(computational task)についても同一の結果を報告すべきことにあり、故に不一致のないことが確認される。インスタンスに故障が存在するときには、この技術は、３つの異なる装置が恐らくは同一のエラーを有していないはずであるという推定を頼りにしており；故に他の２つのインスタンスの多数出力が正当な出力とされ、そして処理のチェーンにおける次のオブジェクトに伝達される。この技術は、ライフサポート(life-support)、ミッションクリティカル(mission critical)、航空宇宙産業、航空産業でしばしば用いられている。これらのタイプのシステムを構築することは、文字通り、少なくとも３回別々にシステムが開発されるのと同様に、明らかに相当に高額である。この技術は、また、しばしばＴＭＲ(triple modular redundancy)と呼ばれる。

【００１２】

信頼性マネージメントを提供するために従来から一般に実施されているエンテティリダンダンシー／レプリケーション方法および他のアプローチは、分散プログラミングネットワークにおける高い有用性を維持することを手助けすることにおいて大きな成功を収めたが、その一方で、それらはある限界を有している。例えば、それらは、欠陥に対する防御に

おける戦略を管理する方法においては概ね静的である。このことは、それらがシステムにおける故障に多くの時間を振り向けることができないことを意味している。それらは、どのコンポーネントがレプリケートされ且つどこにレプリケートされるのかを調整するために人間の介入または制御を必要とする。さらに、シダンダンシー／レプリケーションは、システム故障に対して防御するためには高価な方法であり、そして、もしコンポーネントの全てのインスタンスが同様の問題に苦しめられれば、完全には成功しないおそれがある。

【発明の開示】

【課題を解決するための手段】

【００１３】

本発明は、分散プログラミングネットワークにおいて信頼性バランシングを実施するための方法であって、サービスの要求を受けるステップと、前記要求されたサービスに関連する少なくともひとつのオブジェクトインスタンスを確認するステップと、前記少なくともひとつのオブジェクトインスタンスと前記要求されたサービスとの間の依存性を確認するデータについて質問するステップと、前記確認された少なくともひとつのオブジェクトインスタンスに関連する少なくともひとつの信頼性メトリックについて質問するステップと、少なくともひとつの信頼性メトリックに基づき何れのオブジェクトインスタンスが最も確かに前記サービスを実行し得るかを決定するステップとを備える。

【発明を実施するための最良の形態】

【００１４】

本発明の代表的な実施形態は、添付の図面を用いて本発明の以下の詳細な説明を考慮することで容易に吟味され且つ理解され、各図面において同一の番号が付された要素は同一のものである。

図１は、本発明の代表的な実施形態に係る分散プログラミングネットワークおよび適応信頼性バランシングシステムを示す。

図２は、故障グループを表す図式的な関係のグループを図示し、この故障グループは、図１に示されたコストエバリュエータにより全体の信頼性量のために評価される。

図３は、自己の信頼性見積もりと共に５つのサービスの表現を示す。

図４は、本発明の代表的な実施形態に係る信頼性バランシング方法を示す。

図５は、本発明の典型的な実施形態に係る欠陥耐性サブシステム(fault tolerance subsystem)を示す。

【００１５】

一般に理解されている静的または非適応的な信頼性バランシングアプローチのひとつの結果として、大規模分散プログラミングネットワークは、分散プログラミングネットワークが性能検証(commission)されるまで、即ち稼動するまで、多くの場合、信頼性特性、例えば、分散プログラミングネットワークに特有の問題を示すことができない。加えて、分散プログラミングネットワークおよび分散プログラミングネットワークコンポーネントは、拡大使用のためにしばしば経時的に変化し、性能の低下及び／又は環境的な変化をもたらす。

【００１６】

さらに、分散プログラミングネットワークおよび分散プログラミングネットワークコンポーネントは、しばしば老化の仕方が異なる。例えば、分散プログラミングネットワークまたは分散プログラミングネットワークコンポーネント内のソフトウェアに関しては、ソフトウェアは、分散プログラミングネットワークが引き渡された後に、特定のアプリケーションにアップグレードまたはカスタマイズされることがある。同じことが、特殊化されたハードウェアコンポーネント、および、避けられない(induced)長時間の使用あるいは関係する偶発的事故によるダメージの結果として置き換えられたポスト性能検証(post-commissioning)であるコンポーネントに対しても当てはまる。性能検証に続く分散プログラミングネットワークの構成を変える原因とは関係なく、分散ネットワークおよび分散プログラミングネットワークコンポーネントが変わることが認識されるべきであり、性能検証の

時またはその前に信頼性特性がテストされた分散プログラミングネットワークとは異なる分散プログラミングネットワークの構成を生じる結果となる。

【００１７】

加えて、信頼できるように立案された分散プログラミングネットワークは、長い故障時間、即ち定義によれば故障が起こるまでの時間を有する。この関係の結果のひとつとして、分散プログラミングおよび分散プログラミングネットワークコンポーネントのメーカ(manufacturers)は、しばしば、分散プログラミングネットワークおよび／または分散プログラミングネットワークコンポーネントの信頼性を特徴づけること、および分散プログラミングネットワークおよび／または分散プログラミングネットワークコンポーネントにおける故障を解決するためのソリューションを提供することにおいて、時間と経験が不足している。

【００１８】

さらに、分散プログラミングネットワークは、しばしば、移動するコンポーネント（即ち、クライアントはその移動に気づくことなくひとつのＣＰＵまたはマシンから他に移動するソフトウェアコンポーネント；この移動はコンポーネントによって提供されるサービスのパフォーマンスおよび／または信頼性特性を変え得る）を有する。このような移動するコンポーネントの利用は、分散プログラミングネットワークのダイナミクス(dynamics)および有用性の絶え間なく変化する展望を作り出す。

【００１９】

従って、本発明の実施形態に係る方法およびシステムは、計量(metering)および計時(timing)コンポーネントの集まりを利用し、このコンポーネントは、稼動中の分散プログラミングネットワークの適合性があり且つ動的な調整を可能とするためのフィードバックを提供する。これらの方法およびシステムは、分散プログラミングネットワークが、パワーおよび分散プログラミングネットワーク故障にわたる有用性メトリックを維持し、分散プログラミングネットワークに含まれるソフトウェアおよび／またはハードウェア資源の累積的な信頼性メトリックを提供することを可能とするメカニズムを提供する。本発明の代表的な実施形態は、分散プログラミングネットワークの継続的なモニタリングを提供し、動的な信頼性バランシングを提供する。

【００２０】

本発明の代表的な実施形態に係るシステムおよび方法によって提供されるユーティリティの一つのエリアは、サービスの配達または提供のための最良の有用性状態を保証する改善されたチャンスが存在するようなサービスの消費者とそれらサービスとを知的に結びつけるための能力に関する。サービスの有用性(availability)は、さまざまな方法で算出される。例えば、サービスの有用性は、平均故障時間(MTTF；Mean Time To Failure)を、ＭＴＴＦと平均修復時間(MTTR；Mean Time To Repair)の和で除算したものとして算出できる（即ち、有用性＝ＭＴＴＦ／（ＭＴＴＦ＋ＭＴＴＲ））。

【００２１】

ＭＴＴＦは、初期の瞬間から次の故障までの時間である。ＭＴＴＦ値は、サービス信頼性の統計的数量化である。ＭＴＴＦは、故障から回復してサービスの実施を回復するための時間である。サービスの実施は、モジュール（例えば、協調して動作する１又は２以上のコンポーネント）または他の特定のレファレンスグラニュアリティ(reference granularity)が動作し、指定されたようにサービスを提供するときに成し遂げられる。ＭＴＴＲ値は、サービスの中断の統計的数量化であり、それは、モジュール（または他の特定のレファレンス粒）の振る舞いが、その特定の振る舞いから外れた時間である。

【００２２】

代表的な実施形態によれば、方法および／またはシステムは、“ライブ(live)”または“リアルタイム(real-time)”のデータを利用する。一つの結果として、代表的な実施形態に係るシステムおよび方法は、リアルタイムまたはリアルタイムに近い方法で分散プログラミングネットワークの特性を変えることへの適合を可能とする。このような能力は、極めて長い時間の間の稼動が期待される分散プログラミングネットワークにおける有用性保

証の信用を著しく改善する。

【００２３】

本発明の代表的な実施形態によれば、適合性のある信頼性バランシングは、互いに信頼性の目標が一致しまたは超えるような分散プログラミングネットワークにおいて、クライアントとサーバソフトウェアコンポーネントとの組み合わせに備えるために分散されたクライアントサーバ分散プログラミングネットワーク環境において実施される。この適合性のある信頼性バランシングを提供するために立案されたシステムおよび方法は、分散プログラミングネットワークの現在の構成と分散プログラミングネットワークにおける過去のコンポーネントとの両方が与えられる最も適切な方法で、分散プログラミングネットワークにおいて適合的に信頼性をバランスさせる能力を提供する。このようなシステムおよび方法は、ヒストリー(history)、および分散プログラミングネットワークおよび／または分散プログラミングネットワークサービスに関する将来の要求の統計学的予測に基づく信頼性バランシングを実行するための適合的な計測と共に、バランシング技術を利用する。

【００２４】

蓄積されたデータは、システムを構成するコンポーネントのパフォーマンスのヒストリカルな見通しである。この情報は、将来のパフォーマンスに関する予測的な仮定を提供しようとする場合に使用できる。例えば、コンポーネントに対するＭＴＴＲは、比較的変化しないように思われる。なぜなら、それは、新たなコンポーネントインスタンスを作り出し且つサービスのためにそれを初期化することに関連した時間に対応するからである。結果として、時間を超えて(over time)、任意の特定コンポーネントに対するＭＴＴＲの平均は、一般に、そのコンポーネントの将来の故障に対する修復期間の予測における使用に対しては相当に確かな数である。一方、ＭＴＴＦは、予測性(predictable)は少なく、より確率論的(stochastic)である。結果として、システムの有用性は、潜在的に動的なＭＴＴＦの結果として変化する。

【００２５】

分散プログラミングネットワークにおける他のコンポーネントは、しばしば分散プログラミングネットワークにおける他のコンポーネントの参加を頼りにしているので、全ての又は参加しているコンポーネントの相当数の蓄積された評価は、分散プログラミングネットワークの信頼性を理解するために必要とされる。結果として、本発明の実施形態に係るシステムおよび方法は、場所(location)、時間(time)、依存性(dependency)、および／または特定の分散プログラミングネットワークに関する信頼性(reliability)を収集する。それから、このデータは、コスト評価の発見的方法(cost evaluation heuristics)により分析される。これら発見的方法のファンクションの出力は、有限の複数の選択が存在する分散プログラミングネットワークにおける要求を扱うために、分散されたコンポーネントの最適のおよび／または最も望ましい選択を提供する。

ユーザが定義したメリットファンクションは、ユーザが定義した制約に基づく“最良の適合”を選択するのに適用できる。このようなユーザが定義したメリットファンクションページ：１０は、ガイダンスのためのファンクションへの入力として目標または制約に基づくパラメータを受け取ることができる。

【００２６】

図１は、本発明の代表的な実施形態に係る分散プログラミングネットワーク１００および適応性のある信頼性バランシングシステムを図示する。図１に示されるように、主要な４つの関係要素：クライアント(client)１１０、オブジェクトレゾルバ(object resolver)１２０、依存性マネージャ(dependency manager)１３０、分散オブジェクトインスタンス(distributed object instances)１４０およびオブジェクトメータ(object meters)１５０がある。

図１は、クライアント１１０がタイプ“Ａ”のサーバを使用することを望むという事実を図示している。分散されたオブジェクトインスタンス１４０の集まりは、例えばコントロールファブリック（例えばローカルエリアネットワーク）１６０を介して接続され、３つのこのようなタイプ“Ａ”オブジェクトインスタンス１４１，１４３，１４５、およびひ

とつのタイプ“Ｂ”オブジェクトインスタンス１４７を提供する。図１は、このシナリオの物理的境界を示すものではない。コントロールファブリック１６０は、例えば、独立に動作しているコンポーネント間で、通信を実施し及び／又はパスを制御するハードウェア、ソフトウェアを備えている。そして、そのコンポーネントは、例えばＣＯＲＢＡフレームワークのＩＩＯＰのような分散プログラミングネットワークにおいて、これらの分散プログラミングネットワークコンポーネント（例えばオブジェクトインスタンス１４０）の冗長の間の通信を考慮する。従って、タイプ“Ａ”オブジェクトインスタンス１４１，１４３，１４５は、１又は２以上のモジュールに含まれ、または、１又は２以上のプロセッシングコンポーネント、例えば１つのシャーシ(chassis)における１又は２以上のカード、１つのシャーシにおける１又は２以上のコンピュータ、１つのコンピュータにける１又は２以上の処理等に配置することができる。

【００２７】

クライアント１１０は、例えば、アプリケーションであり、または潜在的に分散されたオブジェクトであり、それは、１又は２以上の分散オブジェクトインスタンス１４０の一つに関連した１又は２以上のサービスの要求された使用のものを捜し又は有するものである。例えば、クライアント１１０は、アプリケーションであり、それはタイプＡ分散オブジェクトインスタンス１４１，１４３，１４５および／またはタイプＢ分散オブジェクトインスタンス１４７において実行されたファンクションまたは方法を呼び出す。本発明の実施形態において、クライアント１１０は、クライアント１１０（図３を参照して後述するように）が期待する信頼性のレベルを示す少なくとも一つの信頼性制約(reliability constraint)を発生し又は割り付けられている。

【００２８】

オブジェクトレゾルバ１２０は、例えば、特定オブジェクトを含むオブジェクトインスタンスと、クライアント１１０によって供給される所望の信頼性制約に適合するオブジェクトのインスタンスとを返すサービスである。依存性マネージャ１３０は、オブジェクト、サービス、またはトポロジーおよび分散オブジェクトインスタンス１４０の間の依存性に関して受容性のあるプロセスである。例えば、依存性マネージャ１３０は、同一のプロセッサまたはプロセッサのセット等を超えて、分散オブジェクトインスタンス１４１および１４３が、同一のコンピュータ上で動作しているか、異なるコンピュータ上で動作しているかを知る。

【００２９】

分散オブジェクトインスタンス１４０は、１又は２以上のクライアント１１０に対するサービスを提供するために使用されるコンポーネントである。分散されたオブジェクトは、オブジェクトとして考えられるが、ネットワーク遠隔機構を通じて、クライアント、例えばクライアント１１０、から遠く離れて（即ち同一のプロセッサ上で稼動せずに）呼び出すことができるという事実によって特徴づけられる。各オブジェクトインスタンス１４０は、プロパティまたは“メータ(meters)”の集まりを備える。これらメータ１５０は、時間に関して累積的である。即ち、コンテンツは、永続的な耐久性のあるストレージ(storage)に保存され、そしてオブジェクトインスタンス１４０が開始される各時間に復元される。

【００３０】

クライアント１１０は、クライアントによって要求された有用性に対する全要求に適合する最良のオブジェクトインスタンスへのレファレンスを得るため、オブジェクトレゾルバ１２０と協議する。オブジェクトレゾルバ１２０は、クライアントに求められたベストマッチを見つけ出すことを試みるため、クライアントに代わってエージェント(agent)またはブローカ(broker)として動作する。もし、このオブジェクトレゾルバが要求を実行できなければ、この実行しだいで、このオブジェクトレゾルバは指示をその結果に返すか、または、おそらく、要求されたパラメータに合致したものを除いてクローゼットマッチ(closet match)を返す。

【００３１】

全ネットワークポリシーは、信頼性ポリシーを含み、例えばオブジェクトレゾルバ１２０に含まれるコストエバリュエータ１２５におけるＸＭＬ(extensible markup language)を通じて断定的に特定される。コストエバリュエータ１２５は、また、オブジェクトインスタンス１４０間の依存性、クライアント１１０の依存性および可能なタイプＡインスタンスの集まりを確認するため、依存性マネージャ１３０を利用する。

【００３２】

分散プログラミングネットワークにおけるオブジェクトまたはサービス間の依存性を確認し且つ理解するための能力は、依存性マネージャ１３０が、故障グループ、即ちサービスのオブジェクトのグループに関する情報を提供することを可能にし、そこでは、構成要素であるオブジェクトまたはサービスのひとつの故障が障害(fault)につながる。その情報は、動的に、または、いくつかの前の断定的な情報（例えば、他の分散プログラミングネットワークコンポーネント、分散プログラミングネットワーク外のコンポーネント、ユーザ管理者、等により決定される）を通じて収集される。この情報は、管理された図表(graph)によって表される。後述のように、この依存性情報は、コストエバリュエータ１２５が、グループの有用性を比較することを可能にする。より大きなグループ（例えば、サービス／オブジェクトおよび／それらの依存したサービス／オブジェクト）は、より低い有用性の格付けを有するようである；故に、それらは、最も高い有用性の測定が必要とされるときに、クライアントとサーバとの間の競争(match)の候補にはなりそうにない。

【００３３】

この依存性情報は、各オブジェクト又はオブジェクトインスタンスが依存していることの一覧を備える。このような一覧は、例えば、図表によって表すことができる。一つの実施例においては、全ての依存性はその一覧で表現される。他の実施例においては、ソフトウェアオブジェクトとクライアントサービスとの間の依存性のみが表現されるのに必要であり；よって、ハードウェアおよび通信依存性は捕獲される必要がない。図２に図示するように、全分散プログラミングが一覧に記入されると、管理された図表の群が結果として生じる。

【００３４】

図２に示すように、群２００（即ち図式的関係２１０）は、図１に図示されたコストエバリュエータ１２５によってそれらの全信頼性の各付けのために評価される故障グループ２１０を表す。各グループ２１０における各オブジェクト／サービス２２０の影響は、簡素化の目的のために等しく取り扱われる；しかしながら、また重み付けされた影響がさらに正確なモデルに適用されることも予測可能である。結果として、本発明の実施形態に係る一つの実施形態において、依存性情報は、グループ２１０のさまざまなオブジェクト／サービスの重要性を示す重み付けされた影響データ(weighted influence data)を含む。これらの故障グループは、概念上、サービス（上述）として考えることができる。

【００３５】

オブジェクトインスタンス１４０間の依存性を確認するデータを受け取った後、コストエバリュエータ１２５は、オブジェクトインスタンスのそれぞれ、例えば１４１，１４３，１４５（さらなる詳細は後述）に関連し、且つクライアントとオブジェクトとの間で滞留しているセッションを実行するためのオブジェクトインスタンスの有用な選択間の例えば相対的コストを決定するのに必要なデータを収集するためにメータ１５０によって提供されるメトリックを評価する。コストエバリュエータ１２５は、それから、信頼性および他のポリシーを適用し、そして“最も適したもの(best fit)”を選択する。

もし、クライアント１１０が、たまたま、コストエバリュエータ１２５に注入されたポリシーに依存して同一のオブジェクトインスタンス１４１，１４３上で動作している場合、信頼性の全体評価が各インスタンス１４１またはインスタンス１４３よりも高いスコアを有しているときにオブジェクトインスタンス１４５に対してレファレンスを返すことがより望ましい。

【００３６】

この評価により提供される情報を用いることなく、従来のシステムは、どのインスタンス

を返すかを決定するためにパフォーマンスバランシングまたは負荷バランシングを単に使用した。これとは対照的に、本発明の実施形態に係るシステムおよび方法は、分散プログラミングネットワークの全体の有用性に関するオブジェクト信頼性の影響が、従来の負荷バランシング技術を通じた性能の最適化と同様に意義深く重要であるという理解に基づいている。

【００３７】

本発明の代表的な実施形態は、部分的には、メータ１５０によって供給されるような信頼性メトリックの継続的蓄積が、信頼性または有用性の決定において役立つという認識に基づいている。この認識のひとつの結果として、さまざまなタイプのデータが、特定ネットワーク全体の有用性のライフタイムの見通しを効果的に計測するために利用される。このデータを収集するため、このシステムおよび方法は、信頼できる方法で時間を超えてデータを収集し、蓄積し、そして固持するための能力を有する。分散プログラミングネットワークの全ライフタイムまたは重要なライフ期間にわたるサービス実行情報の蓄積は、分散プログラミングネットワーク全体の有用性の評定について責任を負う発見的方法に対する意味のある且つより正確な入力を提供する。

【００３８】

個々の分散された各オブジェクトのために収集され且つ蓄積された信頼性メトリックデータのタイプは、例えば、滞在時間（即ち特定サービスが動作していた時間量）、サービス実行時間（即ち特定サービスがファンクショナル（例えばそのファンクションを確かに提供できる）である時間量、そして開始時間（即ち、特定のサービスが、サービスを提供できるようになるための“コールドブート”からスタートするのに要する時間の量；簡略化の目的のため、このメトリックは分散プログラミングネットワークのライフタイムにわたるランニングアベレージ）を含む。加えて、累積するシステム時間は、分散プログラミングネットワークシステム全体が動作している全体時間を示すために記録される。

【００３９】

これらの累積された測定結果を記録することにより、各サービスの信頼性のより正確な理解が提供される。なぜなら、理想的に任意の個々のサービス信頼性が高く、故にＭＴＴＦが理想的に低く、全てのサービスの発生、スタートアップ、またはシステムリセットの後にそれらのカウンタを“リセット”することは大事ではないからである。逆に、これらデータタイプの長期間の累積メトリックによって提供される情報に重要な価値がある。

オブジェクトおよびサービスに蓄積された信頼性メトリックは、オブジェクトレゾルバ１２０におけるコストエバリュエータ１２５に戻される。これは、どのような多くの方法、例えば、サービスの新たな使用に対する要求に基づくオンデマンドで信頼性メトリックを回収(retrieving)するような多くの方法によっても実行できる。

【００４０】

クライアント１１０がサービスの使用を要求した場合、オブジェクトレゾルバ１２０は最初に、サービス、例えばオブジェクトインスタンス１４１，１４３，１４５に対応するサービス、に有用な要求されたタイプの全インスタンスの集まりを確認する。オブジェクトレゾルバ１２０は、例示されたオブジェクトまたはサービスの全てのディレクトリを備えるか又はアクセスすると推定される。一旦候補インスタンスの集まりが確認されると、依存性マネージャ１３０は、オブジェクトとサービスとの間の依存性を確認するデータを確認するために参照される。オブジェクトレゾルバ１２０は、それから質問し、または別に、各インスタンスから信頼性メトリックを順番に回収し、パフォーマンスの改善のために同じ質問から既に訪れたオブジェクトをキャッシング(caching)する。

【００４１】

一旦、信頼性メトリックの全てが収集されると、次のステップで、過去のパフォーマンスが与えられたこのグループ全体の信頼性を確認するための計算を実行する。

要求されたサービスを実行するグループのそれぞれの将来のコスト、例えば費やされた資源の量を計算した後、コストエバリュエータ１２５は、それから、グループのそれぞれを他と比較し、そして格付け(ranking)を行う。この各付けは、コストエバリュエータ１２

５に導入された信頼性評価ポリシーに基づく。
【００４２】
例として、図３は、５つのサービス３１０，３２０，３３０，３４０，３５０を図示し、それぞれは、それら自身の信頼性各付けＲ１－Ｒ５を有しており、その各付けは故障グループ３００の一部である。これら信頼性各付けのそれぞれは、ＭＴＴＦに換算して特定される。それらの信頼性は、１／ＭＴＴＦとして特定される。メータ１５０（図１に示される）によって提供されたオブジェクトメトリックは、同様に有用性の良好な評価を提供する。そのオブジェクトメトリックカウンタから導き出された有用性は、単純に（滞在時間）－（サービス実行時間）である。ＭＴＴＲは、スタートアップタイムのローリングアベレージオブジェクトメトリック(rolling average object metric)であり、それは、コールドスタートからサービス可能となるまでに要する時間の総量を表す。
【００４３】
分散プログラミングネットワークの有用性は、概念上、所要時間に対するサービス実行の比率として定量化でき、例えば、有用性は、統計学上、ＭＴＴＦ／（ＭＴＴＦ＋ＭＴＴＲ）として定量化される。それから、グループ有用性は次のようである：
【００４４】
【数１】

$$a = \frac{\prod_{j=1}^{N} \alpha_j}{\sum_{j=1}^{N} \alpha_j}$$

【００４５】
ここで、αｊは、グループにおける各サービスの有用性を表す。コストエバリュエータ１２５は、各グループに対してこのファンクションを実行し、そしてコストエバリュエータ１２５において特定された信頼性ポリシー（例えば、ポリシーおよび基準）に基づき、最も適切なグループを選択する。例えば、ひとつのポリシーは、特定の信頼性の目標に最も近い信頼性値を有するオブジェクトのグループが、最善のまたは最も信頼できるオブジェクトのグループに対立するものとして常に選ばれる。
【００４６】
図４は、上述した信頼性バランシングの方法を図示している。図４に示されるように、その方法は、ステップ４００で開始して制御がステップ４１０に進む。ステップ４１０では、サービスに対するクライアントの要求が、分散プログラミングネットワークに受け取られる。それから、制御はステップ４２０に進み、そこで、オブジェクトレゾルバが、要求されたサービスに関連するオブジェクトインスタンスを確認する。それから、制御がステップ４３０に進み、そこで、オブジェクトレゾルバが、オブジェクトインスタンスとサービスとの間の依存性を確認するデータについて依存性マネージャに質問する。それから、制御がステップ４４０に進み、そこで、オブジェクトレゾルバは、その関連した信頼性メトリックについて各オブジェクト／サービスに質問する。一旦、各故障グループまたはセットについてのメトリックが回収されると、有用性を評価する次のステップが検討される。それから、制御がステップ４５０に進み、そこでは、信頼性メトリック、依存性、およびコストエバリュエータに含まれ又はアクセスされた信頼性ポリシーに基づき、何れのオブジェクトインスタンスまたはオブジェクトインスタンスのグループが、最も確かにクライアントのサービス要求を実行するかについて決定される。それから、制御がステップ４６０に進み、そこで、この決定は他の分散プログラミングネットワークコンポーネントによって使用され、図５を参照して後述するように、それは、クライアントサービス要求を、選択されたオブジェクトまたはオブジェクトのグループとマッチ(match)させる。それから、制御がステップ４７０に進み、そこで、その方法が終了する。
【００４７】

本発明の実施形態に係る方法およびシステムは、例えば、ＣＯＲＢＡに基づく通信サービスシステムアーキテクチャにおいて実施される。
ＣＯＲＢＡを使用したホステッドサービス(hosted service)を提供するシステムのための或る分散プログラミングネットワークアーキテクチャの一つの利益は、サービスのクライアントが、資源同一のプロセス、同一のホスト、エンベデッドカード、またはネットワークを介して接続された他のマシンにおいて稼動しているかどうかを知らず、また注意も払わないということである。このモデルはそれらの特色を完全に抽出している。このアーキテクチャのひとつの結果は、分散プログラミングネットワークによって提供される全てのサービスおよび資源が通信プロトコル（例えば、ＧＩＯＰに基づく）を通じて緩やかに結合されているので、これらのサービスのクライアント、資源およびＣＯＲＢＡオブジェクトがどのハードウェアと通信しているのかの情報を持たないということである。

【００４８】
本発明の実施形態に係る方法およびシステムは、分散されたオブジェクトモデルに係る分散プログラミングネットワークにおいて使用できる。ＣＯＲＢＡにオブジェクトを配置するための全ての標準的なメカニズムは、このような分散プログラミングネットワークアーキテクチャに適合する。加えて、分散プログラミングネットワークアーキテクチャは、パフォーマンスおよび信頼性の拡張の手助けとなるいくつかの特定ファンクションを実行するためのファンクショナリティ(functionality)を広げることができる。このような分散プログラミングネットワークアーキテクチャには、例えば２つのオブジェクトロケータ(object locators)が存在し、例えば一つは、標準のＩＮＳ(Interoperable Naming Service)であり、もう一つは、図１に図示されるオブジェクトレゾルバ１２０のようなシステム特有のオブジェクトレゾルバである。オブジェクトレゾルバ１２０は、分散プログラミングネットワークにおける信頼性およびパフォーマンスポリシーに基づき自動オブジェクトレファレンスレゾリューション(automatic object reference resolution)を提供するというそのタスクを実行するため、他のコンポーネントと協調してＩＮＳを使用できる。

【００４９】
ＩＮＳは、サービスネーム(service names)をオブジェクトレファレンス(object reference)にマッピング(mapping)するための保管場所を提供する。それは、クライアントが、その特定の場所の情報を要求することなく、ネームでサービスの場所を突き止めることを容易にすることができる。このアーキテクチャを用いて、クライアントは、単純にＩＮＳに質問し、そしてインボケーション(invocation)のために使用できるオブジェクトレファレンスを返す。オブジェクトレファレンスツリーの群(forest of object reference trees)がＩＮＳに置かれ、その例が図２に示される。結果として、依存性マネージャ１３０がＩＮＳを含み、またはＩＮＳに含まれることが理解される。

【００５０】
ＣＯＲＢＡモデルにおける欠陥耐性(fault tolerance)を含むのに必要とされる変化の殆どは、ＩＩＯＰプロトコルに拡張し、新たなＣＯＲＢＡオブジェクトサービスを加えることである。上述したコンポーネントは、それらを分散されたプロセッシングネットワーク内で欠陥耐性サブシステム５００として実施することにより、このような欠陥耐性システムに組み込まれる。結果として、上記の確認されたコンポーネントおよび方法の実施は、ＣＯＲＢＡ欠陥耐性のインフラストラクチャ(infrastructure)をさらに独立的にするネットワークアーキテクチャに組み込まれる。

【００５１】
図５に示されるように、このような欠陥耐性サブシステム５００は、レプリケーションマネージャ５１０、フォールトノティファイア(fault notifier)５２０、少なくとも一つのフォールトディテクタ(fault detector)５３０およびアダプティブプレイサ(adaptive placer)５４０を備え、それはシステム特有のコンポーネントである。このような欠陥耐性サブシステム５００は、さまざまなサービス、例えばレプリケーションマネージャ５１０（例えば欠陥耐性インフラストラクチャにおける管理上のファンクションの殆ど、およびこのサービスのクライアントによって定義された欠陥耐性ドメインのためのプロパティお

よびオブジェクトグループマネージメントを実行すること）、アダプティブプレイサ５４０（例えば、パフォーマンスおよび信頼性ポリシーに基づくオブジェクトレファレンスを作ること）、フォールトノティファイア５２０（例えば、フォールトディテクタおよび／またはこのサービスに登録された消費者にイベントをフィルタリングし且つ伝えるための欠陥通知ハブとして動作すること）、フォールトディテクタ５３０（例えば、レプリケーションマネージャから質問を受け取り、それらの管理下にあるオブジェクトの健康状態をモニタすること）を含む。レプリケーションマネージャ５１０は、欠陥耐性インフラストラクチャの働き者である。

【００５２】

本発明の実施形態に係る欠陥耐性の分散プログラミングネットワークに基づくシステムには、ホスティングサービス(hosting service)に対する複数の候補が存在する。アダプティブプレイサ５４０は、これらの適当な候補を、パフォーマンスおよび信頼性属性(reliability attribute)を有する重み付けされた図表、例えば図1に図示されるオブジェクトメータ１５０によって提供されるメトリック、として具体化する。アダプティブプレイサ５４０は、クライアント、例えば図1に図示されるクライアントに対するアクセスポイントであり、システム特有の特徴と協調してより高い抽出レベルを提供する。アダプティブプレイサ５４０は、各オブジェクトインスタンスの場所を含むデータを生成する。それから、オブジェクトインスタンスまたはオブジェクトグループパフォーマンス（即ちロードバランシング）および信頼性（即ち、信頼性バランシング）係数に基づくクライアントの要求を実行するためのオブジェクトインスタンスを決定するのは、アダプティブプレイサ５４０におけるコストエバリュエーション発見的方法（図２に図示されるオブジェクトレゾルバにおけるコストエバリュエータ１２５に含まれ、それぞれは図５に図示されるアダプティブスペイサ５４０に含まれる）である。

【００５３】

フォールトノティファイア５２０は、1又は２以上のフォールトディテクタ５３０に対するハブとして動作する。フォールトノティファイア５２０は、レプリケーションマネージャ５１０に送る前にフォールトディテクタの通知を収集し、かつ登録された“フォールトアナライザ”をチェックするために使用される。そして、フォールトノティファイア５２０は、アダプティブプレイサ５４０に信頼性メトリックを提供する。

【００５４】

フォールトディテクタ５３０は、単純に、オブジェクトサービスであり、このオブジェクトサービスは、レプリケーションマネージャ５１０により認識されたオブジェクトグループに登録されたオブジェクトの故障を確認するための過酷な努力をしてフレームワークに広がる。フォールトディテクタは、任意のサイズの分散プログラミングネットワークを受け入れるためにヒエラルキー法でスケール(scale)できる。フォールトディテクタ５３０は、図1に図示されるオブジェクトメータ１５０を含んでも良く、これに含まれても良く、またはこれを実行してもよい。

【００５５】

本発明は、上述した特定の実施形態のアウトラインに関連づけて述べられ、多くの代替、修正および変形は当業者に明らかである。従って、上述したように、本発明の代表的な実施形態は例証であって、これに限定されるものではない。本発明の精神および意図から始めることなく、さまざまな変更が可能である。

【図面の簡単な説明】

【００５６】

【図１】本発明の代表的な実施形態に係る分散プログラミングネットワークおよび適応信頼性バランシングシステムを示す図である。

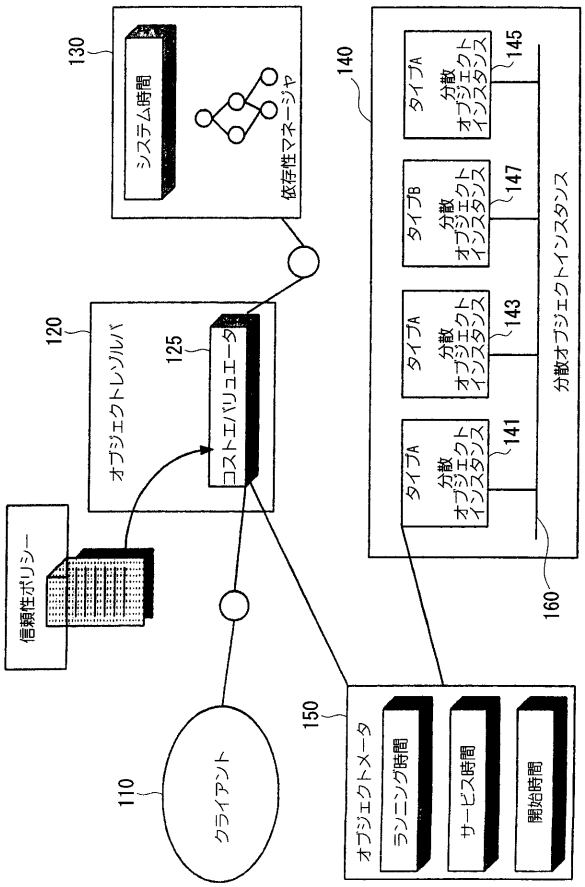
【図２】故障グループを表す図式的な関係のグループを示す図である。

【図３】自己の信頼性見積もりと共に５つのサービスの表現を示す図である。

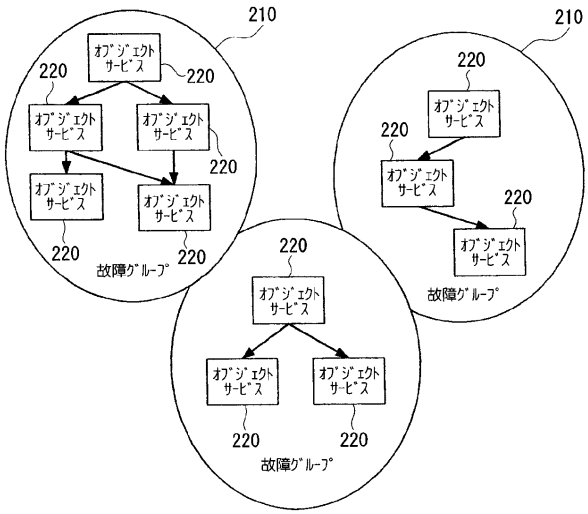
【図４】本発明の代表的な実施形態に係る信頼性バランシング方法を示すフロー図である。

【図５】本発明の代表的な実施形態に係る欠陥耐性サブシステムを示す図である。
【符号の説明】
【００５７】
１１０　クライアント
１２０　オブジェクトレゾルバ
１２５　コストエバリュエータ
１３０　依存性マネージャ
１４０　分散オブジェクトインスタンス
１５０　オブジェクトメータ
２１０　故障グループ
２２０　オブジェクトサービス
５００　欠陥耐性サブシステム
５１０　レプリケーションマネージャ
５２０　フォールトノティファイア
５３０　フォールトディテクタ
５４０　アダプティブプレイサ

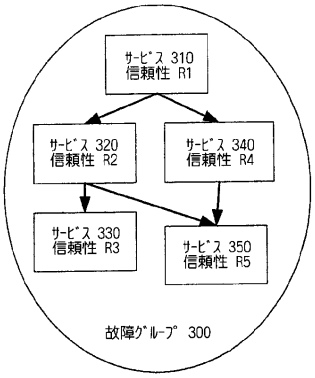
【図１】



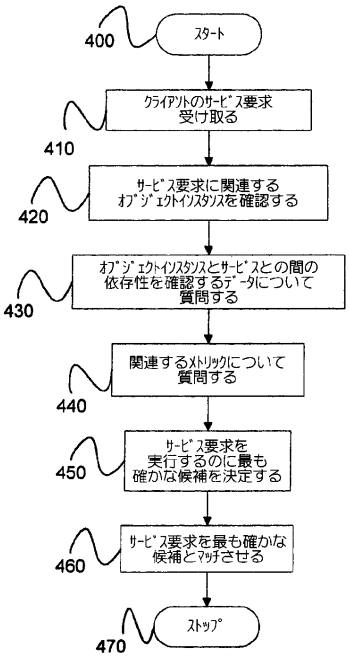
【図２】



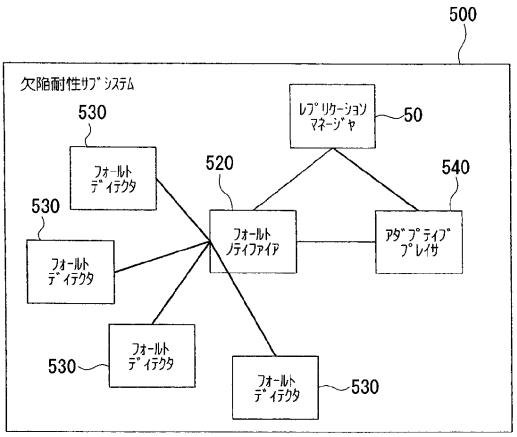
【図３】


ｻｰﾋﾞｽ 310
信頼性 R1
ｻｰﾋﾞｽ 320
信頼性 R2
ｻｰﾋﾞｽ 340
信頼性 R4
ｻｰﾋﾞｽ 330
信頼性 R3
ｻｰﾋﾞｽ 350
信頼性 R5
故障ｸﾞﾙｰﾌﾟ 300

【図４】


400
ｽﾀｰﾄ
ｸﾗｲｱﾝﾄのｻｰﾋﾞｽ要求
受け取る
410
ｻｰﾋﾞｽ要求に関連する
ｵﾌﾞｼﾞｪｸﾄｲﾝｽﾀﾝｽを確認する
420
ｵﾌﾞｼﾞｪｸﾄｲﾝｽﾀﾝｽとｻｰﾋﾞｽとの間の
依存性を確認するﾃﾞｰﾀについて
質問する
430
関連するﾒﾄﾘｯｸについて
質問する
440
ｻｰﾋﾞｽ要求を
実行するのに最も
確かな候補を決定する
450
ｻｰﾋﾞｽ要求を最も確かな
候補とﾏｯﾁさせる
460
ｽﾄｯﾌﾟ
470

【図５】


500
欠陥耐性ｻﾌﾞｼｽﾃﾑ
530
ﾌｫｰﾙﾄ
ﾃﾞｨﾃｸﾀ
ﾚﾌﾟﾘｹｰｼｮﾝ
ﾏﾈｰｼﾞｬ
50
520
540
530
ﾌｫｰﾙﾄ
ﾉﾃｨﾌｧｲｱ
ｱﾀﾞﾌﾟﾃｨﾌﾞ
ﾌﾟﾚｲｻ
ﾌｫｰﾙﾄ
ﾃﾞｨﾃｸﾀ
530
ﾌｫｰﾙﾄ
ﾃﾞｨﾃｸﾀ
ﾌｫｰﾙﾄ
ﾃﾞｨﾃｸﾀ
530

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2002 (04.07.2002)

PCT

(10) International Publication Number
WO 02/052403 A2

(51) International Patent Classification⁷: G06F 9/00

(21) International Application Number: PCT/US01/43640

(22) International Filing Date:
13 November 2001 (13.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/741,869 22 December 2000 (22.12.2000) US

(71) Applicant *(for all designated States except US)*: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **STONE, Alan, E.** [US/US]; 31 Knollwood Drive, Morristown, NJ 07024 (US).

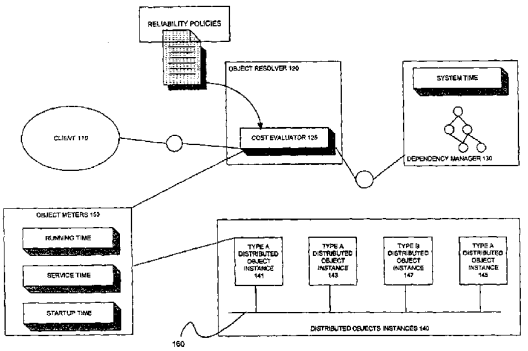
(74) Agents: **WISE, Roger, R.** et al.; Pillsbury Winthrop, Suite 2800, 725 South Figueroa Street, Los Angeles, CA 90017 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR ADAPTIVE RELIABILITY BALANCING IN DISTRIBUTED PROGRAMMING NETWORKS



(57) Abstract: Exemplary embodiments of the invention provide methods and systems for performing reliability balancing, based on past distributed programming network component history, which balances computing resources and their processing components for the purpose of improving the availability and reliability of these resources.

WO 02/052403 A2

WO 02/052403 A2

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR ADAPTIVE RELIABILITY BALANCING IN DISTRIBUTED PROGRAMMING NETWORKS

1. Field of the Invention

The present invention is related to reliability balancing in distributed programming networks. More specifically, the present invention is related to reliability balancing in distributed programming networks based on past distributed programming network and/or distributed programming network component history.

2. Background of the Invention

Computing prior to low-cost computer power on the desktop, was organized in centralized logical areas. Although these centers still exist, large and small enterprises over time are distributing applications and data to where they can operate most efficiently in the enterprise, to some mix of desktop workstations, local area network servers, regional servers, web servers and other servers. In a distributed programming network model, computing is said to be "distributed" when the computer programming and data that computers work on are spread out over more than one computer, usually over a network.

Client-server computing is simply the view that a client machine or application can provide certain capabilities for a user and request others from other machines or applications that provide services for the client machines or applications.

Today, major software makers are fostering an object-oriented view of distributed computing. As a distributed publishing environment with Java and other products that help companies create distributed applications, the World Wide Web is accelerating the trend toward distributed computing. Distributed software models also lend themselves well to provide scalable, highly available systems for large capacity or mission critical systems.

The Common Object Request Broker Architecture (CORBA) is an architecture and specification standard for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker." The International Organization for Standardization (ISO) has sanctioned CORBA as the standard architecture for distributed objects (which are also known as network components).

The essential concept in CORBA is the Object Request Broker (ORB). ORB support in a network of clients and servers on different computers means that a client program (which may itself be an object) can request services from a server program or object without regard for its physical location or its implementation. In CORBA, the ORB is the software that acts as a "broker" between a client request for a service (e.g., a collection of cohesive software functions that together present a server-like capability to multiple clients; services may be, for example, remotely invokable by its clients) from a distributed object or component and the completion of that request. In this way, network components can find out about each other and exchange interface information as they are running. To make requests or return replies between the ORBs, a General Inter-ORB Protocol (GIOP) and, for the Internet, its Internet Inter-ORB Protocol (IIOP) are used. The IIOP maps GIOP requests and replies to the Internet's Transmission Control Protocol (TCP) layer in each computer.

Regardless of what framework or architecture is used in a distributed programming network, the first step in object-oriented programming is to identify all the objects utilized in a system to manipulate and how they relate to each other, an exercise often known as data modeling. Once an object has been identified, the identity of the object is generalized as a class of objects, the type of data it contains and any logic sequences that can manipulate the data are defined. A real instance of a class is called an "object" or, in some

environments, an "instance of a class." For load balancing and reliability balancing (explained herein), multiple instances of the same object may be run at various points within a distributed programming network.

There are two primary challenges to managing large-scale distributed programming systems. One is to maintain high levels of performance when demand for distributed programming network services is high. This challenge is often referred to as "load balancing" and requires balancing the allocation of a finite amount of distributed programming network resources (associated with the distributed programming network services) to larger than usual number of client requests. Often, large-scale distributed programming networks service large numbers of clients of its services. The statistical balancing of this load demand is both a commonly observed and well studied phenomena.

The other primary challenge is maintaining continuous operation of these large-scale distributed programming networks. This challenge may be referred to as "reliability balancing". It is a well understood principal that larger-scale system are more likely to have faults, i.e., causes of service errors. Additionally, the larger the system, the more likely it is that faults will have more significant effects on the consumers of its services. For example, if a service requires resources that utilize or access more than one object, then a failure in any one of these objects may result in a system failure.

Conventionally, there are many approaches for solving load balancing issues of large-scale distributed programming networks. While none of them seem perfect, they have been effective in demonstrating their benefits. However, the challenge of providing distributed programming network reliability, i.e., maintaining operation of distributed programming network management, has matured less rapidly. Conventional methods and distributed programming networks for providing large-scale distributed programming network reliability vary. There are several primary techniques.

The most popular technique for providing large-scale distributed programming network reliability depends on entity, or object instance redundancy. This technique is also often referred to as "replication" and provides some degree of protection against component failures in large or critical systems by providing alternate instances of the same object or group of objects in hopes that when a primary instance of the object or group of objects fails, one or more alternate instances can resume service where the primary left off.

Another common technique, called N-version programming, relies on three or more different versions (implementation) of the same service (or object) running concurrently. Their operation is controlled through some lock-step controlling mechanism such that each of the parallel implementations run logically through the same sequencing without one proceeding ahead of the other for instance. At opportune points in time, the outputs of each of the three or more instances is voted upon. The expectation is that all three instances would report the same results for whatever computational task they are providing, hence no discrepancies should be identified. When there is a failure in an instance, this technique relies upon the presumption that the three different implementations would not likely have the same error; hence, the majority output of the other two instances is taken as the valid output and propagated to the next objects in the chain of processing. This technique is often used in life-support, mission critical, aerospace, and aviation. It is obviously quite expensive to build these types of systems as, literally, the system is developed differently at least three times. This technique is also often called triple modular redundancy (TMR).

While conventionally practiced methods of entity redundancy/replication and other approaches for providing reliability management have been largely successful in helping maintain high availability in distributed programming networks, they have some

limitations. For example, they are largely static in the way that they manage their strategy in the protection against faults. This means they are unable to trend the failures in a system over time. They require human intervention or control to adjust which components are replicated and where they are replicated to. Moreover, redundancy/replication may be a costly way to protect against system faults and may not be entirely successful if all instances of a component suffer from a similar problem.

BRIEF DESCRIPTION OF THE DRAWINGS

The exemplary embodiments of the present invention will be readily appreciated and understood from consideration of the following detailed description of the invention, when taken with the accompanying drawings, in which same numbered elements are identical and:

Fig. 1 illustrates a distributed programming network and components of an adaptive reliability balancing system designed in accordance with an exemplary embodiment of the invention;

Fig. 2 illustrates groups of graphical relationships representing failure groups that may be evaluated for their overall reliability ratings by the cost evaluator illustrated in Fig. 1 ;

Fig. 3 illustrates representations of five services each with their own reliability rating;

Fig. 4 illustrates a method for reliability balancing in accordance an exemplary embodiment of the invention; and

Fig. 5 illustrates a fault tolerance subsystem designed in accordance with an exemplary embodiment of the invention.

DETAILED DESCRIPTION

As one result of the conventionally understood static or non-adaptive reliability balancing approach, large-scale distributed programming networks often fail to exhibit reliability characteristics, for example, problems, particular to the distributed programming networks, until the distributed programming networks are commissioned, i.e., in operation. Additionally, distributed programming networks and distributed programming network components often change over time because of extended use, resulting deterioration and/or environmental changes.

Moreover, distributed programming networks and distributed programming network components often age in different ways. For example, with regard to software within distributed programming networks or distributed programming network components, software may be upgraded or customized to a particular application after a distributed programming network has been commissioned. The same is true for specialized hardware components and components that are replaced post-commissioning as a result of damage, be it long-term use induced or incident related. Regardless of the cause for altering a distributed programming network configuration subsequent to commissioning, it should be appreciated that distributed programming networks and distributed programming network components may change, resulting in a distributed programming network configuration that is different from the distributed programming network configuration that was tested for reliability characteristics at the time of, or prior to, commissioning.

Additionally, distributed programming networks designed to be reliable, have long failure-times, i.e., times in which a failure will occur, by definition. As one result of this relationship, distributed programming network and distributed programming network component manufacturers often have limited time and experience in characterizing the

reliability characteristics of the distributed programming network and/or distributed programming network components and providing solutions for resolving failures in the distributed programming network and/or distributed programming network components.

Further, distributed programming networks often have migratory component (i.e., software components that are able to migrate from one CPU or machine to another, without a client knowing about its migration; this migration may alter the performance and/or reliability attributes of the service provided by the component). Utilization of such migratory components creates an ever-changing view of a distributed programming network's dynamics and availability.

Accordingly, the methods and systems designed in accordance with the exemplary embodiments of the invention utilize a collection of metering and timing components that provide feedback to allow for the adaptive and dynamic calibration of a running distributed programming network. These methods and systems provide a mechanism that allows a distributed programming network to retain availability metrics across power and distributed programming network failures to provide cumulative reliability metrics of software and/or hardware resources included in the distributed programming network. Exemplary embodiments of the invention may provide continuous monitoring of a distributed programming network to provide dynamic reliability balancing.

One area of utility provided by systems and methods designed in accordance with exemplary embodiments of the invention relates to the ability to intelligently couple services and the consumers of those services such that there is an improved chance of assuring the best availability conditions for delivery or provisioning of services. The availability of a service may be calculated in various ways. For example, the availability of a service may be calculated as the Mean Time To Failure (MTTF) divided by the sum

of the MTTF and the Mean Time To Repair (MTTR) (i.e., availability = MTTF/(MTTF+MTTR)).

The MTTF is the time from an initial instant to the next failure event. An MTTF value is the statistical quantification of service reliability. The MTTR is the time to recover from a failure and to restore service accomplishment. Service accomplishment is achieved when a module (e.g., one or more components working in cooperation) or other specified reference granularity acts and provides a service as specified. An MTTR value is the statistical quantification of a service interruption, which is when a module's (or other specified reference granularity) behavior deviates from its specified behavior.

In accordance with an exemplary embodiment, a method and/or system may utilize "live" or "real-time" data. As one result, the systems and methods designed in accordance with that exemplary embodiment may enable adaptation to changing characteristics of a distributed programming network in a real-time or near real-time manner. Such a capability may significantly improve a confidence of availability assurance in distributed programming networks that are expected to run for very long periods of time.

In accordance with an exemplary embodiment of the invention, adaptive reliability balancing may be performed in a distributed, client-server distributed programming network environment to provide for the pairing of a client and server software components in a distributed programming network such that each of them can meet or exceed their reliability goals. Systems and methods designed to provide this adaptive reliability balancing may provide the ability to adaptively balance the reliability in a distributed programming network in a way that is most appropriate given both the present configuration of the distributed programming network and the history of the components in the distributed programming network. Such systems and methods utilize balancing techniques with adaptive measures to perform reliability balancing based on the history

and/or statistical prediction of future demand on the distributed programming network and/or distributed programming network services. Page: 9

The data accumulated is a historical perspective of the performance of the components participating in the system. That information may be used to try to provide predictive assumptions regarding future performance. For example, the MTTR for a component is likely to be relatively invariant because it corresponds to the time associated with creating a new component instance and initializing it for service. As a result, over time, the average of the MTTR for any specific component is generally a fairly confident number for use in the prediction of the repair interval for future failures of that component. The MTTF, on the other hand is likely to be less predictable and more stochastic. As a result, the availability of a system may change as a result of the potentially dynamic MTTF.

Because the components in a distributed programming network often rely on the participation of other components in the distributed programming network, the collective evaluation of all or a substantial number of the components participating is required to understand the reliability of the distributed programming network. As a result, systems and methods designed in accordance with an exemplary embodiment of the invention gather location, time, dependency, and/or reliability data relating to a particular distributed programming network. This data may then be analyzed by cost evaluation heuristics. The output of these heuristic functions may provide an optimal and/or most optimal choice of a distributed component to handle a request in a distributed programming network where there are a finite multiple of choices.

A user defined merit function may be applied to select a "best fit" based on user-defined constraints. Such a user defined merit functionPage: 10 can receive goal or constraint-based parameters as inputs to the function for guidance.

Figure 1 illustrates a distributed programming network 100 and components of an adaptive reliability balancing system designed in accordance with an exemplary embodiment of the invention. As shown in Fig. 1, there are four primary participants: a client 110, an object resolver 120, a dependency manager 130, distributed object instances 140 and object meters 150.

FIG. 1 illustrates the fact that the client 110 may wish to use a service of type 'A'. The collection of distributed object instances 140, e.g., connected via a control fabric (e.g., a local area network) 160, may offer three such type "A" object instances 141, 143, 145 and one type "B" object instance 147. Fig. 1 does not illustrate the physical boundaries of this scenario. The control fabric 160 may include, for example, hardware and software that implement communication and/or control paths between independently running components, which allow for the communication between the redundancy of these distributed programming network components (e.g., the object instances 140) in the distributed programming network 100, e.g., the IIOP of the CORBA framework. Hence, type A object instances 141, 143, and 145 may be included in one or more modules or located on one or more processing components, e.g., one or more cards in a chassis, one or more computers in one chassis, one or more processes in one computer, etc.

The client 110 may be, for example, an application or potentially a distributed object that seeks or has requested use of one or more services associated with one of one or more of the distributed object instances 140. For example, the client 110 may be an application that calls a function or method implemented in the type A distributed object instances 141, 143, 145 and/or the type B distributed object instance 147. In an

embodiment of the invention, the client 110 generates or is assigned at least one reliability constraint that indicates the level of reliability expected by the client 110 (as explained below with reference to Figure 3).

The object resolver 120 may be, for example, a service that returns an object reference indicating a particular object and instance of that object that meets the desired reliability constraints provided by the client 110.

The dependency manager 130 may be an object, service, or process that is knowledgeable regarding the topology and dependencies between the distributed object instances 140. For example, the dependency manager 130 may know that distributed object instances 141 and 143 are running on the same computer, are running on different computers, across the same processor or set of processors etc.

The distributed object instances 140 may be components that are used to provide services for one or more clients 110. A distributed object may be thought of as an object but characterized by the fact that the object is remotely (i.e., not running on the same processor) invokable from a client, e.g., client 110, through a network remoting mechanism. Each object instance 140 has a collection of properties or "meters". These meters 150 may be cumulative over time. That is, the contents may be preserved in persistent and durable storage, then reinstated each time the object instance 140 is started.

The client 110 may confer with the object resolver 120 to obtain a reference to the optimal object instance 140 that meets the overall requirements for availability requested by the client. The object resolver 120 acts as an agent or broker on behalf of the client to try to find the best match requested of the client. If the object resolver is unable to fulfill the request, depending on the implementation, the object resolver may either return an indication to that effect or perhaps return the closet match short of meeting the requested parameters.

The overall network policies, including reliability policies, may be specified declaratively, e.g., through eXtensible Markup Language (XML) in the cost evaluator 125 included in the object resolver 120. The cost evaluator 125 may also utilize the dependency manager 130 to identify dependencies between the object instances 140, dependencies of the client 110 and the collection of possible type A instances.

The ability to identify and understand the dependencies between objects or services in the distributed programming network allows the dependency manager 130 to provide information regarding failure groups, i.e., groups of objects or services, in which failure of one of the constituent objects or services may lead to a fault. The information may be gathered dynamically, or through some prior declarative information (e.g., determined by another distributed programming network component, a component outside the distributed programming network, a user or administrator, etc.). The information may be represented by a directed graph. As explained below, this dependency information allows the cost evaluator 125 to compute the availability of a group. Larger groups (e.g., services/objects and their dependent services/objects) will likely have lower availability ratings; hence, they may be less likely candidates for a match between a client and a server when the highest availability measures are needed.

This dependency information may include an inventory of what each object or object instance is dependent on. Such an inventory could be represented, for example, by a graph. In one implementation, all dependencies may be depicted in the inventory. In another implementation, only the dependency between the software objects and client services is necessary to be depicted; thus, hardware and communications dependencies need not be captured. As illustrated in Figure 2, when an entire distributed programming network is inventoried, a forest of directed graphs may result.

As shown in Fig. 2, the forest 200 (i.e., groups of graphical relationships 210) represents failure groups 210 that may be evaluated for their overall reliability ratings by the cost evaluator 125 illustrated in Fig. 1. The influence of each object/service 220 in each group 210 may be treated equivalently for simplicities sake; however, it is foreseeable also that the math for weighted influences may also be applied for a more accurate model. As a result, in one implementation of an exemplary embodiment of the invention, the dependency information may include weighted influence data that indicates the significance of various objects/services 220 of groups 210. It should be appreciated that these failure groups may be conceptually thought of as services (described above).

After receiving the data identifying the dependencies between the object instances 140, the cost evaluator 125 may evaluate the metrics associated with each of the object instances, e.g., 141, 143, 145 (explained in more detail below) and provided by the meters 150 to gather the necessary data to determine, for example, relative costs between the available choices of object instances to fulfill abinding session between the client and the object. The cost evaluator 125 may then apply the reliability and other policies, and select a "best fit".

If the client 110 happens to be running on the same object instances 141 and 143, depending on the policy injected into the cost evaluator 125, it may be more desirable to return a reference to object instance 145 if the overall evaluation of reliability has a higher score than either instance 141 or 143.

Without the information provided by this evaluation, conventional systems merely used performance balancing or load balancing to determine which object instance to return. In contrast, systems and methods designed in accordance with the exemplary embodiments of the invention are based on an understanding that influence of object reliability on the overall availability of a distributed programming network is as significant

and important as optimization of performance through conventional load balancing techniques.

The exemplary embodiments of the invention are based, in part, on a recognition that persistent accumulation of reliability metrics such as those provided by the meters 150 may be valuable in performing a reliability or availability determination. As one result of this recognition, various types of data may be utilized to effectively measure a lifetime view of a particular network's overall availability. To gather this data, the systems and methods have the ability to collect, accumulate, and persist this data over time in a reliable manner. The accumulation of service accomplishment information over the full lifetime or a significant period of the life of the distributed programming network helps provide meaningful and more accurate input into the heuristics that are responsible for an assessment of the overall distributed programming network availability.

Types of reliability metrics data that may be collected and accumulated for each individual distributed object may include, for example, sojourn time (i.e., the amount of time a particular service has been operating), service accomplishment time (i.e., the amount of time a particular service has been functional (e.g., able to provide its functions reliably)), and startup time (i.e., the amount of time it takes a particular service to start from a "cold boot" to being able to provide service; for simplicities sake, this metric may be a running average over the lifetime of the distributed programming network.) In addition, cumulative system time may be recorded to indicate an overall time the entire distributed programming network system has been running.

By recording these cumulative measurements, a more accurate understanding of the reliability of each service may be provided. Because, ideally, the reliability of any individual service is high, and hence the MTTF is ideally low, it is not valuable to "reset" these counters after every service creation, startup, or system reset. To the contrary, there

is significant value in the information provided by these data types' long-term cumulative metrics.

The reliability metrics accumulated in the objects and services may be communicated back to the cost evaluator 125 in the object resolver 120. This may be accomplished any number of ways, for example, retrieving the reliability metrics on demand based upon requests for new use of a service.

When a client 110 requests the use of a service, the object resolver 120 first identifies the collection of all instances of the requested type available for service, e.g., service A corresponds to object instances 141, 143 and 145. The object resolver 120 is presumed to either include or have access to a directory of all the instantiated objects or services. Once a collection of candidate instances has been identified, the dependency manager 130 is consulted to identify the data identifying the dependencies between the objects and services. The object resolver 120 then queries or otherwise retrieves the reliability metrics from each object instance in turn, caching already visited objects from the same query for performance improvements.

Once all of the reliability metrics have been collected, the next step is to now perform some calculations to identify the overall availability of this group given its past performance.

After calculating the prospective cost, e.g., amount of resources expended, of each of the groups fulfilling the service request, the cost evaluator 125 then compares each of the groups to one another and performs a ranking. This ranking is based upon the reliability evaluation policies injected into the cost evaluator 125.

As an example, Figure 3 illustrates five services 310, 320, 330, 340 and 350, each with their own reliability rating R1-R5 that are part of a failure group 300. Each of these reliability ratings may be specified in terms of the MTTF. Their reliability may then be

specified as 1/MTTF. The object metrics provided by the meters 150 (illustrated in Fig. 1) may provide a good estimate of the availability as well. The availability derived from the object metrics counters is simply the (sojourn time) - (the service accomplishment time). The MTTR may be the rolling average object metric of the startup time, which may represent the amount of time required to go from a cold start to serviceability.

The availability of a distributed programming network may be conceptually quantified as the ratio of the service accomplishment to the elapsed time, e.g., the availability is statistically quantified as: MTTF / (MTTF + MTTR). The group availability is then the following:

$$a = \frac{\prod_{j=1}^{N} \alpha_j}{\sum_{j=1}^{N} \alpha_j}$$

where α_j represents the availability of each service in the group. The cost evaluator 125 may perform this function for each group, then, select the most appropriate group based on reliability policies (e.g., policies and criteria) specified in the cost evaluator 125. For example, one policy may be that the group of objects having a reliability value that is closest to the specified reliability goal is always chosen as opposed to the best or most reliable group of objects.

Figure 4 illustrates a method for reliability balancing in accordance with the above-description. As shown in Fig. 4, the method begins at 400 and control proceeds to 410. At 410, a client's request for service is received by the distributed programming network. Control then proceeds to 420, at which the object resolver identifies the object instances associated with the requested service. Control then proceeds to 430, at which the object resolver queries the dependency manager for data identifying the dependencies between the objects instances and services. Control then proceeds to 440, at which the object

resolver queries each object/service for its associated reliability metrics. Once the metrics for each failure group or set has been retrieved, the next step of evaluating the availability is considered. Control then proceeds to 450, at which a determination is made as to which object instance or group of object instances may most reliably fulfill the client's service request, based on the reliability metrics, dependencies and reliability policies included in or accessed by the cost evaluator. Control then proceeds to 460, at which this determination may be used by other distributed programming network components, as explained in relation to Figure 5 below, that matches the client service request with the selected object or group of objects. Control then proceeds to 470, at which the method ends.

Methods and systems designed in accordance with the exemplary embodiments of the invention may be implemented, for example, in a subsystem that may be a CORBA-based, communication services system architecture.

One benefit of some distributed programming network architectures for systems providing hosted services using CORBA is that clients of the services may not know, nor care, whether or not resources are running in the same process, same host, an embedded card, or another machine connected via a network. The model entirely abstracts these particulars. One consequence of this architecture, because all services and resources provided by the distributed programming network are loosely coupled through a communications protocol (e.g., based on GIOP), the clients of these services, resources and CORBA objects have no knowledge of what hardware they are communicating with.

The methods and systems designed in accordance with the exemplary embodiments of the invention may be used in a distributed programming network designed in accordance with a distributed object model. All the standard mechanisms for locating objects in CORBA may apply in such a distributed programming network

architecture. In addition, the distributed programming network architecture may extend the functionality to perform some specific functions that aid in performance and reliability scalability. In such a distributed programming network architecture, there may be, for example, two object locators, e.g., one that may be a standard Interoperable Naming Service (INS) and another that may be a system-specific object resolver such as object resolver 120 illustrated in Fig. 1. The object resolver 120 may use the INS along with other components to perform its task of providing automatic object reference resolution based on reliability and performance policies in the distributed programming network.

The INS may provide a repository for mapping service names to object references, which makes it easy for a client to locate a service by name without requiring knowledge of its specific location. With this architecture, a client can simply query the INS and have returned an object reference that can then be used for invocations. Located in the INS is a forest of object reference trees, an example of which is shown in Fig. 2. As a result, it should be appreciated that the dependency manager 130 may include or be included in the INS.

Most of the changes needed to include fault tolerance in a CORBA model are enhancements to the IIOP protocol and the addition of a few new CORBA object services. The components described above may be incorporated in such a fault tolerant system by implementing them as a fault tolerance subsystem 500 within a distributed processing network. As a result, the above-identified components and method operations may be incorporated in a network architecture that makes a CORBA fault tolerance infrastructure more autonomous.

As shown in Figure 5, such a fault tolerance subsystem 500 may include a replication manager 510, fault notifier 520, at least one fault detector 530 and an adaptive placer 540, which is a system-specific component. Such a fault tolerance subsystem 500

may contain various services, e.g., those associated with the replication manager 510 (e.g., performing most of the administrative functions in the fault tolerance infrastructure and the property and object group management for fault tolerance domains defined by the clients of this service), the adaptive placer 540 (e.g., creating object references based on performance and reliability policies), the fault notifier 520 (e.g., acting as a failure notification hub for fault detectors and/or filtering and propagating events to consumers registered with this service), and the fault detector 530 (e.g., receiving queries from the replication manager, monitoring the health of objects under their supervision, etc.). The replication manager 510 is the workhorse of the fault tolerance infrastructure.

In a fault tolerant, distributed programming network based system designed in accordance with the exemplary embodiments of the invention, there are multiple candidates for hosting services. The adaptive placer 540 models these eligible candidates as a weighted graph that has performance and reliability attributes, e.g., the metrics provided by the object meters 150 illustrated in Fig. 1. The adaptive placer 540 may be the access point for the client, e.g., for client 110 illustrated in Fig. 1, providing a higher level of abstraction along with some system-specific features. The adaptive placer 540 may create data indicating the location of each object instance. It is then the cost evaluation heuristics (included in the cost evaluator 125 in the object resolver 120 illustrated in Fig. 2 each included in the adaptive placer 540 illustrated in Fig. 5) in the adaptive placer 540 that determines the best object instance to fulfill a client request based on object instance or object group performance (i.e., load balancing) and reliability (i.e., reliability balancing) coefficients.

The fault notifier 520 may act as a hub for one or more fault detectors 530. The fault notifier 520 may be used collect fault detector notifications and check with registered

"fault analyzers" before forwarding them on to the replication manager 510. Thus the fault notifier 520 may provide the reliability metrics to the adaptive placer 540.

The fault detectors 530 are simply object services that permeate the framework in a relentless effort to identify failures of the objects registered in the object groups recognized by the replication manager 510. Fault detectors can scale in a hierarchical manner to accommodate distributed programming networks of any size. It should be appreciated that the fault detectors 530 may include, be included in or implement the object meters 150 illustrated in Fig. 1.

While this invention has been described in conjunction with the specific embodiments outlined above, it is evident that many alternatives, modifications and variations will be apparent to those skilled in the art. Accordingly, the exemplary embodiments of the invention, as set forth above, are intended to be illustrative, not limiting. Various changes may be made without departing from the spirit and scope of the invention.

We claim:

1. A method for performing reliability balancing in a distributed programming network, the method comprising:

receiving a service request;

identifying at least one object instance associated with the requested service;

querying for data identifying dependencies between the at least one object instance and the requested service;

querying for at least one reliability metric associated with the identified at least one object instance; and

determining which object instance will most reliably fulfill the service request based on that at least one reliability metric.

2. The method of claim 1, wherein determining which object instance will more reliably fulfill the service request is also based on dependencies between the at least one object instance and the requested service.

3. The method of claim 1, wherein determining which object instance will more reliably fulfill the service request is also based on reliability policies of the distributed programming network.

4. The method of claim 1, further comprising matching the service request with at least one object instance based on the determination of which object instance will most reliably fulfill the service request based on the at least one reliability metric.

5. The method of claim 4, wherein matching the service request comprises evaluating at least one reliability metric corresponding to at least one of a history and statistical prediction of future service demand on object instances included in the distributed programming network.

6. A system configured to perform reliability balancing in an operating distributed programming network, the system comprising:

an object resolver configured to identify at least one object instance associated with a requested service from a plurality of object instances coupled together via a control fabric, to query for at least one reliability metric associated with the identified at least one object instance and to make a determination as to which object instance will most reliably fulfill the service request;

a dependency manager coupled to the object resolver, the dependency manager being configured to provide data identifying dependencies between the at least one object instance and the requested service; and

at least one object meter configured to generate the at least one reliability metric regarding at least one object instance.

7. The system of claim 6, wherein the object resolver includes a cost evaluator that has access to reliability policies specific to the distributed programming network.

8. The system of claim 6, wherein the system is configured to retain availability metrics across power and system failures to provide cumulative reliability metrics corresponding to objects and object instances within the distributed programming network.

9. The system of claim 6, wherein the system performs continuous monitoring of the distributed programming network to provide dynamic reliability balancing.

10. The system of claim 6, wherein the system performs matching between service requests and objects to fulfill the service requests by evaluating the availability of at least one object instance to provide the requested service.

11. The system of claim 10, wherein the availability of the object instance is calculated based on a mean time to failure and a mean time to repair.

12. The system of claim 10, wherein the availability of the object instance is calculated as a mean time to failure divided by the sum of the mean time to failure and the mean time to repair.

13. The system of claim 12, wherein the mean time to failure is a time period from an initial instant to a next failure event.

14. The system of claim 13, wherein the mean time to failure is a statistical quantification of system service reliability.

15. The system of claim 12, wherein the mean time to failure is the time to recover from a failure and to restore service accomplishment.

16. The system of claim 15, wherein service accomplishment is achieved when objects working in cooperation to provide the requested service provide the requested service as specified.

17. The system of claim 12, wherein the mean time to repair is a statistical quantification of a service interruption.

18. The system of claim 6, wherein the object resolver evaluates real-time data regarding the operation of at least one object instance or group of object instances.

19. The system of claim 18, wherein the system enables adaptation of service request routing based on changing characteristics of the distributed programming network.

20. The system of claim 19, wherein adaptation is performed in real-time.

21. The system of claim 6, wherein the service request originates from an application or a distributed programming network object that seeks or has requested use one of one or more of the distributed objects.

22. The system of claim 6, wherein service request originates from a client, which generates or is assigned at least one reliability constraint that indicates a level of reliability expected by that client.

23. The system of claim 12, wherein the object resolver is a service that returns reference identification data indicating a particular object and instance of that object that meets the at least one reliability constraint provided by the client.

24. The system of claim 6, wherein the object resolver is a service that returns reference identification data indicating a particular object and instance of that object that meets the at least one reliability constraint provided in the service request.

25. The system of claim 6, wherein the dependency manager is a service that is knowledgeable regarding the topology and dependencies between distributed object instances included in the distributed programming network.

26. The system of claim 6, wherein the object resolver generates a reference to an optimal object instance that meets overall distributed programming network requirements.

27. The system of claim 26, wherein the overall distributed programming network requirements includes at least one reliability policy.

28. The system of claim 6, wherein the data identifying dependencies includes an inventory of what each object or object instance is dependent on.

29. The system of claim 6, wherein the at least one object meter generates at least one reliability metric that is cumulative over time.

30. The system of claim 6, wherein the at least one reliability metric includes or is based on a service sojourn time.

31. The system of claim 6, wherein the at least one reliability metric includes or is based on a service accomplishment time.

32. The system of claim 6, wherein the at least one reliability metric includes or is based on a startup time.

33. A fault tolerance subsystem for improving fault tolerance in a distributed programming network, the fault tolerance subsystem comprising:

a replication manager configured to perform object group management within a distributed programming network including a dependency manager being configured to provide data identifying dependencies between at least one object instance and a requested service;

at least one fault detector configured to receive and respond to queries from the replication manager and monitor a status of objects and object instances within the distributed programming network under the at least one fault detector's supervision and configured to generate the at least one reliability metric regarding at least one object instance within the distributed programming network;

a fault notifier coupled to the replication manager and the at least on fault detector and configured to act as a failure notification hub for the at least one fault detector by notifying the replication manager of object or object instance failure following receipt of data indicating detection of such a fault from the at least one fault detector; and

an adaptive placer configured to identify at least one object instance associated with a requested service from a plurality of object instances, to query for at least one reliability metric associated with the identified at least one object instance and to make a determination as to which object instance will most reliably fulfill the service request.

34. The fault tolerance subsystem of claim 33, wherein the object resolver includes a cost evaluator that has access to reliability policies specific to the distributed programming network.

35. The fault tolerance subsystem of claim 33, wherein the service request originates from a client, which generates or is assigned at least one reliability constraint that indicates a level of reliability expected by that client.

36. The fault tolerance subsystem of claim 35, wherein the object resolver is a service that returns reference identification data indicating a particular object and instance of that object that meets the at least one reliability constraint provided by the client.

37. The fault tolerance subsystem of claim 35, wherein the dependency manager is a service that is knowledgeable regarding the topology and dependencies between distributed object instances included in the distributed programming network.

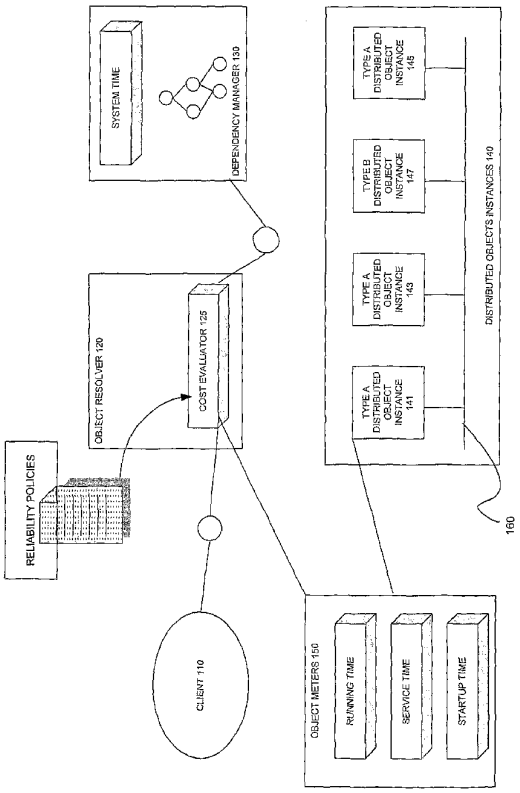

SYSTEM TIME
DEPENDENCY MANAGER 130
OBJECT RESOLVER 120
COST EVALUATOR 125
RELIABILITY POLICIES
CLIENT 110
OBJECT METERS 150
RUNNING TIME
SERVICE TIME
STARTUP TIME
TYPE A DISTRIBUTED OBJECT INSTANCE 141
TYPE A DISTRIBUTED OBJECT INSTANCE 143
TYPE B DISTRIBUTED OBJECT INSTANCE 147
TYPE A DISTRIBUTED OBJECT INSTANCE 145
DISTRIBUTED OBJECTS INSTANCES 140
160

FIGURE 1

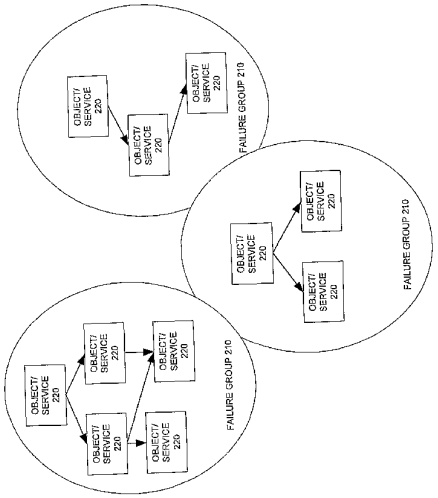

OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
FAILURE GROUP 210
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
FAILURE GROUP 210
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
OBJECT/ SERVICE 220
FAILURE GROUP 210

FIGURE 2

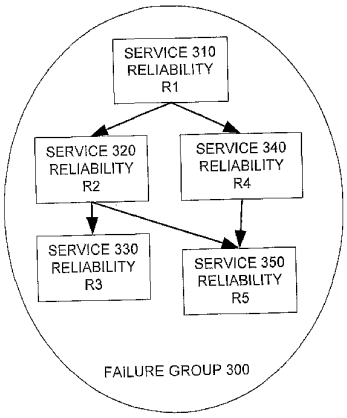

SERVICE 310
RELIABILITY
R1
SERVICE 320
RELIABILITY
R2
SERVICE 340
RELIABILITY
R4
SERVICE 330
RELIABILITY
R3
SERVICE 350
RELIABILITY
R5
FAILURE GROUP 300

FIGURE 3

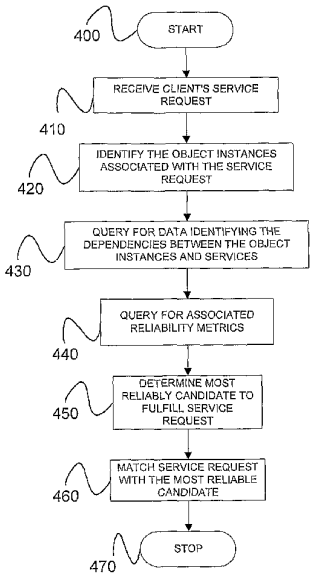

400
START
RECEIVE CLIENT'S SERVICE REQUEST
410
IDENTIFY THE OBJECT INSTANCES ASSOCIATED WITH THE SERVICE REQUEST
420
QUERY FOR DATA IDENTIFYING THE DEPENDENCIES BETWEEN THE OBJECT INSTANCES AND SERVICES
430
QUERY FOR ASSOCIATED RELIABILITY METRICS
440
DETERMINE MOST RELIABLY CANDIDATE TO FULFILL SERVICE REQUEST
450
MATCH SERVICE REQUEST WITH THE MOST RELIABLE CANDIDATE
460
STOP
470

FIGURE 4

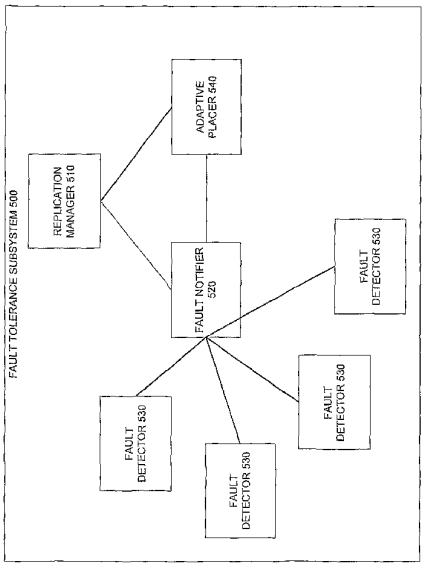

FAULT TOLERANCE SUBSYSTEM 500
REPLICATION MANAGER 510
ADAPTIVE PLACER 540
FAULT NOTIFIER 520
FAULT DETECTOR 630
FAULT DETECTOR 530
FAULT DETECTOR 530
FAULT DETECTOR 530
FAULT DETECTOR 530

FIGURE 5

【国際公開パンフレット（コレクトバージョン）】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2002 (04.07.2002)

PCT

(10) International Publication Number
WO 02/052403 A3

(51) International Patent Classification[7]: G06F 11/00, H04L 12/24

(21) International Application Number: PCT/US01/43640

(22) International Filing Date:
13 November 2001 (13.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/741,869 22 December 2000 (22.12.2000) US

(71) Applicant *(for all designated States except US)*: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **STONE, Alan, E.** [US/US]; 31 Knollwood Drive, Morristown, NJ 07024 (US).

(74) Agents: **WISE, Roger, R.** et al.; Pillsbury Winthrop, Suite 2800, 725 South Figueroa Street, Los Angeles, CA 90017 (US).

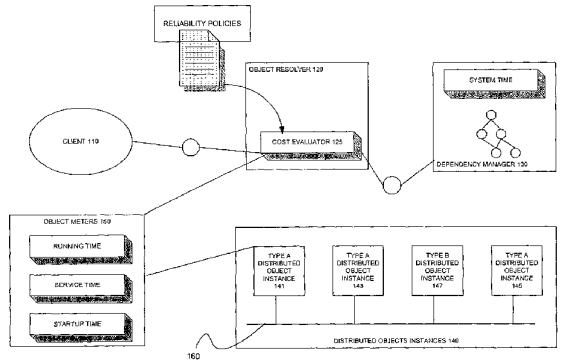
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR ADAPTIVE RELIABILITY BALANCING IN DISTRIBUTED PROGRAMMING NETWORKS



(57) Abstract: Exemplary embodiments of the invention provide methods and systems for performing reliability balancing, based on past distributed programming network component history, which balances computing resources and their processing components for the purpose of improving the availability and reliability of these resources.

WO 02/052403 A3

WO 02/052403 A3

(88) Date of publication of the international search report:
9 January 2003

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/43640

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/00 H04L12/24

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 99 38095 A (TELENOR AS) 29 July 1999 (1999-07-29) abstract page 1, line 15 - line 23 page 5, line 29 -page 6, line 2	1,6,33
A	C. SABNIS ET AL.: "Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQua" DEPENDABLE COMPUTING FOR CRITICAL APPLICATIONS, 6 January 1999 (1999-01-06), pages 149-168, XP010366445 San Jose, CA, USA the whole document	1,6,33

-/--

[X] Further documents are listed in the continuation of box C. [X] Patent family members are listed in annex.

* Special categories of cited documents :
A document defining the general state of the art which is not considered to be of particular relevance
E earlier document but published on or after the international filing date
L document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
O document referring to an oral disclosure, use, exhibition or other means
P document published prior to the international filing date but later than the priority date claimed

T later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
X document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
Y document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
& document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
29 July 2002	23/08/2002
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Absalom, R

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/43640

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	J. KOISTINEN: "Dimensions for Reliability Contracts in Distributed Object Systems" HEWLETT PACKARD TECHNICAL REPORT, 3 October 1997 (1997-10-03), pages 1-34, XP002207748 H-P Labs the whole document	1-33
A	S. FROLUND ET AL.: "Quality of Service Specification in Distributed Object Systems Design" 4TH USENIX CONF ON OBJECT-ORIENTED TECHNOLOGIES AND SYSTEMS, 27 April 1998 (1998-04-27), XP002207749 Santa Fe, New Mexico, USA the whole document	1-33
A	J. ORVALHO ET AL.: "Augmented Reliable Multicast CORBA Event Service (ARMS): A QoS-Adaptive Middleware" PROC. OF 7TH INT WKSP ON INTERACTIVE DISTRIBUTED MULTIMEDIA SYSTEMS, 17 October 2000 (2000-10-17), XP008005865 Enschede, The Netherlands the whole document	1-33

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 01/43640

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9938095 A	29-07-1999	US 5983225 A	09-11-1999
		US 6044370 A	28-03-2000
		AU 3278999 A	09-08-1999
		EP 0990214 A2	05-04-2000
		JP 2001523367 T	20-11-2001
		WO 9938095 A1	29-07-1999

Form PCT/ISA/210 (patent family annex) (July 1998)

フロントページの続き

(51)Int.Cl.[7]	ＦＩ	テーマコード（参考）
	Ｇ０６Ｆ　9/06　６２０Ｈ	

(81)指定国　AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,OM,PH,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,US,UZ,VN,YU,ZA,ZW

(74)代理人　100107836
弁理士　西　和哉
(74)代理人　100108453
弁理士　村山　靖彦
(74)代理人　100110364
弁理士　実広　信哉
(72)発明者　アラン・イー・ストーン
アメリカ合衆国・ニュージャージー・０７０２４・モリスタウン・ノルウッド・ドライヴ・３１