(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4781089号
(P4781089)

(45)発行日　平成23年9月28日(2011.9.28)　(24)登録日　平成23年7月15日(2011.7.15)

(51)Int.Cl.　　F I
G06F 9/50 (2006.01)　　G06F 9/46 465Z
G06F 9/48 (2006.01)　　G06F 9/46 452F

請求項の数 12　(全 31 頁)

(21)出願番号　特願2005-330887(P2005-330887)
(22)出願日　平成17年11月15日(2005.11.15)
(65)公開番号　特開2007-140710(P2007-140710A)
(43)公開日　平成19年6月7日(2007.6.7)
審査請求日　平成20年11月13日(2008.11.13)

(73)特許権者　310021766
株式会社ソニー・コンピュータエンタテインメント
東京都港区港南１丁目７番１号
(74)代理人　100105924
弁理士　森下　賢樹
(74)代理人　100109047
弁理士　村田　雄祐
(74)代理人　100109081
弁理士　三木　友由
(74)代理人　100134256
弁理士　青木　武司

最終頁に続く

(54)【発明の名称】タスク割り当て方法およびタスク割り当て装置

(57)【特許請求の範囲】
【請求項１】
それぞれがプロセッサを備え相互に通信可能に接続された複数のノードを含む分散処理システムにおいて、アプリケーションに含まれる先後関係を有する複数のタスクを複数のノード間での計算結果の整合性が維持されるように各ノードに割り当てるタスク割り当て方法であって、
タスク割り当ての処理を実行するノードのプロセッサが、
前記アプリケーションを解析して、または予め定められた、タスク間の先後関係、各タスクに科せられる時間制約、タスク処理時間、タスク間転送データ量、ノード間通信スループットを取得し、
タスク毎に、当該タスクと先後関係にある全ての先行タスクについて、先行タスクの処理開始時刻と、先行タスクの処理時間と、当該タスクと先行タスクの間のタスク間転送データ量をノード間通信スループットで除して求めた通信時間とを加えた時刻を算出し、算出された時刻のうち最も遅い時刻を、当該タスクを最も早く処理開始可能な時刻である最先開始時刻と決定し、
タスク毎に、当該タスクと先後関係にある全ての後続タスクについて、後続タスクの処理開始時刻から、当該タスクの処理時間と当該タスクと後続タスクの間のタスク間転送データ量をノード間通信スループットで除して求めた通信時間とを減じた時刻を算出し、算出された時刻のうち最も早い時刻を、当該タスクの処理を終了するための最も遅い開始時刻である最遅開始時刻と決定し、

前記最遅開始時刻から前記最先開始時刻を減じたタスク可動範囲を算出し、
少なくとも一部のタスクについて、タスクが割り当てられるべきノードの事前指定を受け付け、
前記少なくとも一部のタスクを事前指定されたノードに優先して割り当てた後、残りのタスクについて、前記タスク可動範囲が小さいタスクから順に割り当て対象タスクと決定し、該割り当て対象タスクの処理時間分の空き時間を有するノードを割当先ノードとして選択し、該割当先ノードに前記割り当て対象タスクを割り当てることを特徴とするタスク割り当て方法。

【請求項２】
前記タスク割り当ての処理を実行するノードのプロセッサが、
前記事前指定がなされていないタスクを、前記複数のノードに実際に割り当てる前に一時的に配置するための仮想ノードに割り当て、
前記仮想ノードに割り当てられたタスクと他のノードに割り当てられたタスクとの間で、前記仮想ノードと他のノードとの間の通信スループットとして予め定められている仮想スループットでタスク間転送データ量を除して求めた仮想データ通信時間を算出し、該仮想データ通信時間を前記通信時間の代わりに使用して前記最先開始時刻および最遅開始時刻を算出することを特徴とする請求項１に記載のタスク割り当て方法。

【請求項３】
前記タスク割り当ての処理を実行するノードのプロセッサが、
前記タスク可動範囲が最小となるタスクが複数ある場合、当該タスクと後続タスクとの間でのタスク間転送データ量を、当該タスクと後続タスクが割り当てられたノード間のノード間通信スループットで除して求めた通信時間が最長となるタスクを選択し、
選択したタスクと後続タスクとを一体的に取り扱うようグループ化し、
グループ単位で前記タスク可動範囲を算出し、
前記タスク可動範囲が小さいグループからグループ単位で割当先ノードを選択し、該割当先ノードに前記割り当て対象タスクを割り当てることを特徴とする請求項１または２に記載のタスク割り当て方法。

【請求項４】
それぞれがプロセッサを備え相互に通信可能に接続された複数のノードを含む分散処理システムにおいて、アプリケーションに含まれる先後関係を有する複数のタスクを複数のノード間での計算結果の整合性が維持されるように各ノードに割り当てるタスク割り当て装置であって、
タスク間の先後関係、各タスクに科せられる時間制約、タスク処理時間、タスク間転送データ量、ノード間通信スループットを取得する情報取得部と、
タスク毎に、当該タスクと先後関係にある全ての先行タスクについて、先行タスクの処理開始時刻と、先行タスクの処理時間と、当該タスクと先行タスクの間のタスク間転送データ量をノード間通信スループットで除して求めた通信時間とを加えた時刻を算出し、算出された時刻のうち最も遅い時刻を、当該タスクを最も早く処理開始可能な時刻である最先開始時刻と決定し、かつ、タスク毎に、当該タスクと先後関係にある全ての後続タスクについて、後続タスクの処理開始時刻から、当該タスクの処理時間と、当該タスクと後続タスクの間のタスク間転送データ量をノード間通信スループットで除して求めた通信時間とを減じた時刻を算出し、算出された時刻のうち最も早い時刻を、当該タスクの処理を終了するための最も遅い開始時刻である最遅開始時刻と決定する、開始時刻算出部と、
少なくとも一部のタスクについて、タスクが割り当てられるべき割当先ノードの指定を受け付ける事前指定受付部と、
前記少なくとも一部のタスクを指定された割当先ノードに優先して割り当てた後、残りのタスクについて前記最遅開始時刻から前記最先開始時刻を減じたタスク可動範囲を算出し、該タスク可動範囲が小さいタスクから順に割り当て対象タスクと決定し、該割り当て対象タスクの処理時間分の空き時間を有するノードを割当先ノードとして選択するノード選択部と、

選択された割当先ノードに前記割り当て対象タスクを配置するタスク配置部と、
を備えることを特徴とするタスク割り当て装置。
【請求項５】
割当先ノードの事前指定がなされていないタスクを、前記複数のノードに実際に割り当てる前に一時的に配置するための仮想ノードに割り当て、前記仮想ノードに割り当てられたタスクと他のノードに割り当てられたタスクとの間で、前記仮想ノードと他のノードとの間の通信スループットとして予め定められている仮想スループットでタスク間転送データ量を除して仮想データ通信時間を算出するタスク間通信時間取得部をさらに備え、
前記開始時刻算出部は、前記仮想データ通信時間を前記通信時間の代わりに使用して前記最先開始時刻および前記最遅開始時刻を算出することを特徴とする請求項４に記載のタスク割り当て装置。
【請求項６】
前記ノード選択部は、複数の割り当て対象タスクについて前記タスク可動範囲が同一の場合、割り当て対象タスクとその後続タスクとの間の前記仮想データ通信時間が大きな割り当て対象タスクについて、割当先ノードを優先的に選択することを特徴とする請求項５に記載のタスク割り当て装置。
【請求項７】
前記ノード選択部は、優先的に割当先ノードが決定された割り当て対象タスクとその後続タスクとを同一ノードに割り当てることを特徴とする請求項６に記載のタスク割り当て装置。
【請求項８】
前記ノード選択部は、複数の割り当て対象タスクについて後続タスクとの間の前記仮想データ通信時間が同一である場合、全ての割り当て対象タスクのうち最先開始時刻が最小であるタスクについて、割当先ノードを優先的に選択することを特徴とする請求項６に記載のタスク割り当て装置。
【請求項９】
前記ノード選択部は、ひとつ以上のタスクが既に割り当てられているノードに対してさらに割り当て対象タスクを割り当てるとき、前記ノードに前記割り当て対象タスクの処理時間分の空き時間があるか否かを判定し、空き時間がないとき、前記ノードに割り当て済みのタスクの最遅開始時刻を移動させることで前記空き時間を作れるか否かを判定する空き時間検出部をさらに含むことを特徴とする請求項４ないし８のいずれかに記載のタスク割り当て装置。
【請求項１０】
前記ノード選択部は、前記タスク可動範囲が最小となる割り当て対象タスクが複数ある場合、割り当て対象タスクと後続タスクとの間でのタスク間転送データ量を、割り当て対象タスクと後続タスクが割り当てられたノード間のノード間通信スループットで除して求めた通信時間が最長となるタスクをグループ化するグループ化部をさらに備え、
前記開始時刻算出部は前記グループを単位として前記最先開始時刻と最遅開始時刻とを算出し、
前記ノード選択部は、前記グループを単位として割当先ノードを選択し、選択した割当先ノードにグループ内の全てのタスクを割り当てることを特徴とする請求項４ないし９のいずれかに記載のタスク割り当て装置。
【請求項１１】
前記事前指定受付部は、複数のタスクをグループ化する指定を受け付け、
前記グループ化部は、指定通りにタスクのグループ化を実行することを特徴とする請求項１０に記載のタスク割り当て装置。
【請求項１２】
それぞれがプロセッサを備え相互に通信可能に接続された複数のノードを含む分散処理システムにおいて、アプリケーションに含まれる先後関係を有する複数のタスクを複数のノード間での計算結果の整合性が維持されるように各ノードに割り当てるタスク割り当て

プログラムであって、
前記アプリケーションを解析して、または予め定められた、タスク間の先後関係、各タスクに科せられる時間制約、タスク処理時間、タスク間転送データ量、ノード間通信スループットを取得する機能と、
タスク毎に、当該タスクと先後関係にある全ての先行タスクについて、先行タスクの処理開始時刻と、先行タスクの処理時間と、当該タスクと先行タスクの間のタスク間転送データ量をノード間通信スループットで除して求めた通信時間とを加えた時刻を算出し、算出された時刻のうち最も遅い時刻を、当該タスクを最も早く処理開始可能な時刻である最先開始時刻と決定する機能と、
タスク毎に、当該タスクと先後関係にある全ての後続タスクについて、後続タスクの処理開始時刻から、当該タスクの処理時間と、当該タスクと後続タスクの間のタスク間転送データ量をノード間通信スループットで除して求めた通信時間とを減じた時刻を算出し、算出された時刻のうち最も早い時刻を、当該タスクの処理を終了するための最も遅い開始時刻である最遅開始時刻と決定する機能と、
前記最遅開始時刻から前記最先開始時刻を減じたタスク可動範囲を算出する機能と、
少なくとも一部のタスクについて、タスクが割り当てられるべきノードの事前指定を受け付ける機能と、
前記少なくとも一部のタスクを事前指定されたノードに優先して割り当てた後、残りのタスクについて、前記タスク可動範囲が小さいタスクから順に割り当て対象タスクと決定し、該割り当て対象タスクの処理時間分の空き時間を有するノードを割当先ノードとして選択し、該割当先ノードに前記割り当て対象タスクを割り当てる機能と、
をタスク割り当ての処理を実行するノードのプロセッサに実現させるためのタスク割り当てプログラム。

【発明の詳細な説明】
【技術分野】
【０００１】
この発明は、プロセッサを有する複数のノードからなる分散処理システムにおいて、各ノードへタスクを割り当てる技術に関する。
【背景技術】
【０００２】
プロセッサを有する複数のノードからなる分散処理システムにおいてアプリケーションを実行するためには、アプリケーションのタスクをいずれのノードで実行すべきか決定する必要がある。この場合、あるイベントの結果が全てのノードに影響を及ぼすような重要な計算について、いかにして整合性を維持するかが問題となる。従来から、イベントをノード間で相互に通信して各ノードで整合性を維持する方式や、代表ノードが専用サーバの役割を果たし、代表ノードでのみ重要なタスクを実行することで整合性を維持する方式が知られている。
【発明の開示】
【発明が解決しようとする課題】
【０００３】
しかしながら、上記第１の方式では、各ノードでの演算結果が相違し整合性がとれなくなる場合がある。また、第２の方式では、専用サーバの役割を果たす代表ノードに他のノードを接続する形態をとることになる。したがって、アプリケーションの実行中に代表ノードを他のノードに変更したり、新たなノードを追加したりすることは困難である。
【０００４】
本発明はこうした課題に鑑みてなされたものであり、その目的は、プロセッサを有する複数のノードからなる分散処理システムにおいて、アプリケーションのタスクをいずれのノードに割り当てるべきかを判定する技術を提供することにある。
【課題を解決するための手段】
【０００５】

本発明のある態様は、それぞれがプロセッサを備え相互に通信可能に接続された複数のノードを含む分散処理システムにおいて、先後関係を有する複数のタスクを各ノードに割り当てるタスク割り当て方法である。この方法では、ひとつまたは複数のプロセッサが、各タスクについて、当該タスクを最も早く処理開始可能な時刻である最先開始時刻と、時間制約内にタスクを終了するための最も遅い開始時刻である最遅開始時刻とを算出し、最先開始時刻と最遅開始時刻との差分であるタスク可動範囲を算出し、タスク可動範囲が小さいタスクから順に割当先ノードを決定する処理を実行する。

【０００６】

ここで、「タスク」とは、ある目的を達成するためにプログラムされたアプリケーションまたはそれに含まれる情報処理の内容をいい、アプリケーションと対応してもよいし、入出力制御やユーザが指定したコマンドなどアプリケーションよりも小さい単位と対応してもよく、何らかの処理または機能の単位に対応すればよい。

この態様によれば、タスク可動範囲に基づいて各タスクの割当先ノードが決定されるので、事前に代表ノードを定めることなく、アプリケーションタスクを処理時間の観点から最適なノードに割り当てることができる。

【０００７】

なお、本発明の構成要素や表現を方法、システム、コンピュータプログラム、コンピュータプログラムを格納した記録媒体などの間で相互に置換したものもまた、本発明の態様として有効である。

【発明の効果】

【０００８】

本発明によれば、分散処理システムにおいて各ノードへのタスクの割り当てが自動的に決定されるので、代表ノードを予め定めておく必要がない。

【発明を実施するための最良の形態】

【０００９】

本発明は、複数のノードで構成される分散処理システムにおいて、アプリケーションを実行する際に、タスクの持つ時間制約を充足できるようにタスクを各ノードに割り当てる技術に関する。

【００１０】

以下、本発明の一実施の形態に係るシステム構成と、当該システムで実行されるタスクの概略を説明した後、各機能ブロックの動作をフローチャートを参照して詳細に説明する。

【００１１】

図１は、本発明の実施の形態に係る分散アプリケーション実行システムの構成図である。各ノード１０は、汎用のコンピュータからなり、それぞれひとつ以上のＣＰＵを備える。各ＣＰＵは、同一の命令セットを使用可能であるとする。各ノード１０は、インターネットを初めとするネットワーク３０を介して相互にデータ送受信可能に構成されている。図１では、ノード１～５の５台のコンピュータがネットワーク３０に接続されているが、分散処理システムを構成するノードの数に制限はない。

図１の環境において、分散アプリケーションが実行される。ここで、「分散アプリケーション」とは、ネットワークで接続された複数のＣＰＵ搭載機器を同時に使用し、分担して処理を進める形のアプリケーションをいう。

【００１２】

図２は、各ノードを構成する汎用コンピュータ１００の概略構成を示す。コンピュータ１００は、ＣＰＵ１２、メインメモリ１４、記憶装置１６、入出力インタフェース２０および表示制御装置２８を備える。ＣＰＵ１２は、コンピュータ１００の全体を制御するとともに、ノードへのタスク割り当てプログラムを実行する。メインメモリ１４は、各種のデータや描画処理プログラムなどを記憶する。これらはバス１８を介して接続され、相互にデータの送受信が可能となっている。表示制御装置２８には、アプリケーション実行の結果生成される画像を出力するディスプレイ２６が接続される。入出力インタフェース２

０には、ＣＤ－ＲＯＭ、ＤＶＤ－ＲＯＭまたはハードウェアディスクドライブなどの外部記憶装置２２と、コンピュータ１００に対してデータを入力するためのキーボードやマウス等の入力装置２４が接続される。入出力インタフェース２０は、外部記憶装置２２および入力装置２４に対するデータの入出力を制御する。入出力インタフェース２０は、外部記憶装置２２に格納されたデータやプログラムを読み込み、メインメモリ１４に提供する。入出力インタフェース２０には、他のノードを構成するコンピュータと通信して、データおよびプログラムを取得する通信装置６０も接続される。

【００１３】

以下の実施形態では、分散アプリケーションとしてネットワーク対戦型の格闘ゲームを想定して説明するが、本実施の形態を適用可能なアプリケーションはこれに限られるわけではなく、複数のノードでタスク処理が必要となる任意の分散アプリケーションに適用することができる。

【００１４】

図３は、上記ゲームアプリケーションにおいて、各プレイヤーに対しての処理を担当するノードで実行されるべきタスク、およびタスク間の実行順序の制約の概略を示す。図中のブロックはそれぞれタスクを表し、矢印はタスク間の実行順序の制約を表す。実線の長方形ブロックは、各プレイヤーの担当ノードでのみ実行されるべきタスクを表す。例えば、ＢＧＭ演奏３１、キー入力３２、背景表示３６、キャラクタ表示３９、効果音発生４０といったタスクは、各ノードでそれぞれのプレイヤーに対して実行されなければ、ゲームアプリケーションとしての意味をなさない。長方形に縦線入りのブロックは、ネットワーク接続される任意のノードで実行可能なタスクを表す。キャラクタ移動・座標計算３３およびダメージ計算３７は、その計算結果を用いた表示が各プレイヤーのノードで出力される限り、実行されるノードを選ばない。

【００１５】

二重囲いのブロックは、全てのプレイヤーについての計算を一カ所のノードで実行すべきタスクを表す。実行されるノードは任意である。衝突判定３４は、ゲーム環境内のキャラクタ間の接触の結果を計算するタスクであるが、このような計算は、複数のノードで実行すると整合がとれなくなる場合があるので、全プレイヤーの座標計算の終了後に一カ所のノードで集中して演算すべきである。さらに、点線の長方形ブロックは、ひとつのノードで計算されてもよいし、分散されて複数のノードで計算されてもよいタスクを表す。ゲーム環境内のキャラクタの移動とは無関係に変化する背景を計算する背景変化計算３５は、その計算結果が各ノードに提供されれば、ひとつのノードで実行してもよい。

【００１６】

図４は、本実施の形態によるタスク割り当て結果の一例を示す。図４中のノード１～５は、図１のノード１～５と対応している。このうち、ノード１はプレイヤー１の処理を担当するノードであり、ノード５はプレイヤー２の処理を担当するノードであり、ノード２～４はプレイヤーの付いていないノードとする。図４では、実線長方形のブロックおよび長方形に縦線入りのブロックのうち、斜線なしのブロックはプレイヤー１に関するタスク、斜線を付したブロックはプレイヤー２に関するタスクを表す。

【００１７】

図４に示すように、キー入力３２、ＢＧＭ演奏３１、背景表示３６といったタスクは、各プレイヤーに対して割り当てられたノード１、ノード５上に割り当てられる。図４では、プレイヤー１に関するキャラクタ移動・座標計算３３は、ノード１ではなくノード２に割り当てられ、プレイヤー２に関するキャラクタ移動・座標計算３３は、ノード５ではなくノード４に割り当てられている。また、一カ所のノードで実行すべき衝突判定３４は、ノード２およびノード４のキャラクタ移動・座標計算３３からそれぞれデータを受け取ってノード３上で実行され、その計算結果がノード１、２、５に割り当てられたタスクに伝達されている。この結果、全てのノード上で計算結果の整合性を保ちつつ、ノード１およびノード５において、音声再生や画面表示が実現される。なお、このタスク割り当ては一例であり、他の割り当て結果も当然存在する。

【００１８】
このように、一口にタスクといっても、特定のノードで処理すべきなのか、または任意のノードで処理してもよいのかなど、条件は様々である。また、タスク毎の処理時間も異なるし、演算結果をタスク間で転送するのに要する時間も異なる。さらに、ゲームアプリケーションは画面の描画を伴うので、１フレーム（例えば、１／６０秒）以内にプレイヤー一人分の処理、すなわちキー入力から画面表示までの一連の処理を終了しなければならない。
したがって、分散アプリケーション実行環境において、アプリケーション内のタスクをいずれのノードに割り当てるかは、全てのノードにおける演算結果の整合性や、リアルタイム性を確保するための処理時間などに大きな影響を及ぼしうる。
以下では、図１に示すような分散アプリケーション実行システムにおいて、アプリケーションのタスクを自動的にかつ適切に分配するタスク割り当て方法について説明する。
【００１９】
図５は、実施の形態に係るタスク割り当て処理を実行するノード１０の機能ブロック図である。各ブロックは、分散アプリケーション実行環境においてタスク割り当てを実現するために必要な処理を実行する。なお、ノード１０は、ハードウェア的には、ＣＰＵ、ＲＡＭ、ＲＯＭなどの構成で実現でき、ソフトウェア的にはデータ入力機能、データ保持機能、演算機能、通信機能などの諸機能を発揮するプログラムで実現されるが、図５ではそれらの連携によって実現される機能ブロックを描いている。したがって、これらの機能ブロックはハードウェア、ソフトウェアの組み合わせによって様々な形で実現できる。
【００２０】
ユーザ入力部１０２は、キーボードやマウス等の入力装置を介してユーザからのデータ入力を受け取る。
情報収集部１０８は、タスク割り当てを決定するために必要となる各種情報を収集する。情報収集部１０８は、タスクに関する情報を取得するタスク情報取得部１１０と、ノードに関する情報を取得するノード情報取得部１１２と、後述する事前指定を受け付ける事前指定受付部１１８とを含む。
【００２１】
タスク情報取得部１１０が取得するタスク情報には、タスク間の先後関係、リアルタイム性を確保するために各タスクに科せられる時間制約、最悪ケースでの各タスクの処理に要する時間であるタスク処理時間、タスク間の転送データ量が含まれる。ここで、時間制約は、例えば、あるアプリケーションを所定の間隔で周期的に繰り返し実行すべきときは、その間隔が時間制約となり得るし、または、あるアプリケーションの実行を開始してから終了するまでの制限時間も制約条件のひとつである。この時間制約は、以下の説明では「デッドライン時刻」として表現されている。
【００２２】
ノード情報取得部１１２が取得するノード情報には、ノードリスト、ノード間通信レイテンシ、ノード間通信スループット、ノードリソース情報が含まれる。このうち、ノードリソース情報は、タスクを割り当てるべき各ノードにおける計算負荷の状態や、各ノードのＣＰＵ処理能力、メモリ容量といった計算リソースに関する情報である。これらの情報は、図示しない負荷監視部が、ネットワークに接続されている各ノードから現時点での負荷量の通知を受けるようにするか、または、オペレーティングシステムにこれらの情報を伝達する仕組みを設けることで取得する。
なお、タスク間の転送データ量と、ノード間のレイテンシ、スループット情報から、タスク間の通信時間を計算することができる。
【００２３】
また、事前指定受付部１１８が受け付ける情報には、タスク割り当てノード事前指定、タスクグループ事前指定が含まれる。
【００２４】
各タスクの時間制約、タスク処理時間、タスク間転送データ量、ノードリスト、ノード

間通信レイテンシ、ノード間通信スループットは、実行されるアプリケーションのプログラマが予め入力しておいたものを使用するか、または、プログラム解析部１０４が、プログラム解析ツールを用いてアプリケーションプログラムを解析して推定した結果を使用してもよい。ノード間通信レイテンシとノード間通信スループットは、ネットワーク構成から推定される推定値を使用してもよい。

【００２５】

格納部１３０は、情報収集部１０８で取得されたタスク割り当て処理に必要となる各種データを格納する。タスク情報格納部１２０は、タスク先後関係、タスク時間制約、タスク処理時間、タスク間転送データ量、タスク割り当てノード事前指定情報、タスクグループ事前指定情報を格納する。ノード情報格納部１２６は、ノードリスト、ノード間通信レイテンシ、ノード間通信スループット、ノードリソース情報を格納する。

【００２６】

タスク割り当て部１４０は、格納部１３０に存在する各種情報のうち少なくともひとつの情報を参照して、ネットワーク内のノードにアプリケーションタスクを割り当てるタスク割り当て処理を実行する。タスク割り当て部１４０は、開始時刻算出部１４４、対象タスク選択部１４６、ノード選択部１４８を含む。

【００２７】

開始時刻算出部１４４は、各タスクまたはタスクグループについて、それぞれ最先開始時刻（ＡＥＳＴ：Absolute Earliest Start Time ）、および最遅開始時刻（ＡＬＳＴ：Absolute Latest Start Time）を計算する。ＡＥＳＴの算出については図７および図８を参照して説明し、ＡＬＳＴの算出については図９および図１０を参照して説明する。

対象タスク選択部１４６は、タスク割り当ての対象となるタスクを選択する。この選択には、上述のＡＥＳＴおよびＡＬＳＴが利用される。この対象タスク選択処理については、図１１のフローチャートを参照して説明する。

ノード選択部１４８は、タスク割り当ての対象となるタスク（以下、「割り当て対象タスク」という）を、ネットワーク内のいずれのノードに割り当てるかのノード選択処理を実行する。この処理については、図１３～図１７のフローチャートを参照して説明する。

【００２８】

タスク配置部１５０は、タスク割り当て部１４０における処理結果にしたがってタスクを各ノードに配置し、タスクを実際に動作させるのに必要な情報、すなわち、タスクのプログラムコード、初期データ、先後関係のあるタスクの割り当てられたノードなどの情報を伝達する。プログラムコードそのものの代わりに、各ノードの記憶装置に予め複数のコードを記録しておき、必要なコードの管理番号などを伝達してもよい。

【００２９】

配置されたタスクは、各ノードで分散処理される。このとき、各ノードがそれぞれ勝手にタスクの実行を開始しても統制がとれなくなる可能性がある。そのため、分散アプリケーションの全タスクの全てが実行可能になるのを待機し、全ノードで一斉にタスクの実行を開始するなどの処理を実現するために、タスク配置部１５０は、実行の開始、中断、停止などの命令を各ノードに発行してもよい。

【００３０】

タスク配置部１５０は、ネットワーク内の各ノードの状況の変化、例えば、新たなノードが追加されたり、またはノードがネットワークから切断されたりして、各タスクの実行が困難になった場合に、タスク割り当て部１４０にタスクの再配置をするように指示してもよい。

【００３１】

続いて、本実施の形態におけるタスク割り当て処理の概要を説明する。

各タスクについて、タスクの最先開始時刻ＡＥＳＴと、デッドライン時刻を守るための最遅開始時刻ＡＬＳＴを求め、その差が最小であるタスク、すなわち最も時間的に余裕の少ないタスクを選択し、このタスクの割当先のノードを決定する。ＡＥＳＴやＡＬＳＴは、タスク処理時間やタスク間の通信時間などを考慮して算出される。ひとつのタスクの割

当先ノードが決定すると、その結果にしたがって全タスクについてＡＥＳＴとＡＬＳＴを再計算し、その計算結果にしたがって次の割り当て対象タスクを決定する。このようにして、重要なタスクから割当先のノードが決定されていく。

【００３２】

図６は、ノード１０で実施されるタスク割り当て処理のメインフローチャートである。

まず、アプリケーションのプログラマは、ノードおよびタスクに関する所定の情報をネットワーク内のいずれかのノードに格納する。この格納先は、タスク割り当て処理を実行するノードに接続された記憶装置であることが好ましいが、ネットワークを介して接続される別のノードの記憶装置であってもよい。格納すべき情報は、上述の時間制約やタスク処理時間である。特定のノードでのみ実行すべきタスクがある場合は、ノード事前指定情報やグループ事前指定情報も格納する。上述の情報を固定値として予め準備しておく代わりに、各ノードがアプリケーションプログラムを解析することによってこれらの情報を求めてもよい。

【００３３】

次に、ネットワーク内のいずれかのノードの情報収集部１０８が、上述のタスク情報、ノード情報、および事前指定情報を取得する（Ｓ１２）。続いて、各ノードにいずれのタスクを割り当てるかを決定していく。このタスク割り当て処理は、五段階に分けて実行される。まず、ノード事前指定のあるタスクについて割り当て処理を実行する（Ｓ１４）。ここでは、単にタスクを指定されたノードに割り当てる。但しこのとき、ノード事前指定のあるタスクにグループ事前指定がある場合は、指定されたノードにグループの全タスクを割り当てる。次に、ノード事前指定のないタスクをそれぞれ別の仮想ノードに割り当てる（Ｓ１６）。但しこのとき、グループ事前指定のあるタスクについては、グループ毎に同一の仮想ノードに割り当てる。続いて、仮想ノードに割り当てられているタスクのうちデッドライン制約のあるタスクについて、図１１を参照して後述するタスク割り当て処理を実行する（Ｓ１８）。さらに、仮想ノードに割り当てられているタスクのうちデッドライン制約のないタスクについて、後述するタスク割り当て処理を実行する（Ｓ２０）。最後に、ノードに割り当てられた全てのタスクの実行時刻を、Ｓ１４～Ｓ２０において計算されるＡＥＳＴに設定した上で、タスク配置部１５０が、それぞれのＡＥＳＴの時刻にタスクが実行されるように各ノードに配置する（Ｓ２２）。

【００３４】

なお、図６のタスク割り当て処理を担当するノードは、分散アプリケーションの実行要求を出すノードとすることが好ましい。したがって、割り当て処理の担当ノードは、アプリケーションの実行要求者全員がなる可能性がある。但し、割り当て処理が同時並行して行われるので、各ノードの負荷状態などがこのタイミングで更新されたときには、必要に応じて割り当て処理の再実行をするように構成しておく。

なお、分散アプリケーション実行システム内のいずれかのノードを、別の方法でタスク割り当て処理を担当するノードに決定してもよい。この場合、割り当て処理を担当するノードを、システム内のいくつかのノードに予め限定しておいてもよい。

【００３５】

ここで、ＡＥＳＴおよびＡＬＳＴについて説明する。

最先開始時刻ＡＥＳＴは、あるタスクを最も早く処理開始可能な時刻を表す。最先開始時刻は以下のように求める。計算対象のタスクの全ての先行タスクについて、その先行タスクの最先開始時刻に、当該先行タスクの処理時間と、当該先行タスクから計算対象タスクまでの通信時間とを加えた時刻を求め、この計算値が最大となる（すなわち、最も時刻の遅い）ものが計算対象タスクのＡＥＳＴとなる。

最遅開始時刻ＡＬＳＴは、あるタスクを時間制約内に終了するための最も遅い開始時刻を表す。最遅開始時刻は以下のように求める。（１）計算対象のタスクにデッドライン制約がある場合は、（１－１）計算対象タスクの全ての後続タスクについて、その後続タスクの最遅開始時刻から、計算対象タスクの処理時間と、計算対象タスクからその後続タスクまでの通信時間とを減じた時刻、（１－２）計算対象タスクのデッドライン時刻から当

該計算対象タスクの処理時間を減じた時刻、のそれぞれを計算し、これらの計算値のうち最小となる（すなわち、最も時刻の早い）ものが計算対象タスクのＡＬＳＴとなる。（２）計算対象のタスクにデッドライン制約がない場合は、計算対象タスクの全ての後続タスクについて、その後続タスクの最遅開始時刻から、計算対象タスクの処理時間と、計算対象タスクからその後続タスクまでの通信時間とを減じた時刻を計算し、この計算値が最小となる（すなわち、最も時刻の早い）ものが計算対象タスクのＡＬＳＴとなる。

【００３６】

ＡＥＳＴ、ＡＬＳＴの演算方法については、Yu-Kwong Kwok, Ishfaq Ahmad, Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, 1996 March, vol. 7, pp. 506-521,に記載されている。本明細書ではその概略のみを説明する。

【００３７】

図７（ａ）は、ＡＥＳＴの算出方法を模式的に示しており、図７（ｂ）は、タスク間の依存関係の一例を示す。図７（ｂ）において、白丸内の数字がタスク番号を表しており、矢印が依存関係を表す。すなわち、図７（ｂ）では、「タスク４」の先行タスクとして「タスク１」、「タスク２」、「タスク３」があることを示している。

ここで、「タスク間の依存関係」とは、あるタスクの処理結果が他のタスクの処理に用いられるような関係をいう。互いに依存関係にあるタスクにおいて、対象タスクより時間的に先行するタスクを「先行タスク（親タスクともいう）」と呼び、時間的に後続するタスクを「後続タスク（子タスクともいう）」と呼ぶ。

【００３８】

任意のノードＪにおけるあるタスクn_iのＡＥＳＴは、次式で算出される。なお、n_iはＡＥＳＴの計算対象となるタスクであり、n_{ik}はタスクn_iのｋ番目の先行タスクであり、$1 \leqq k \leqq p$、すなわちn_iの先行タスクがｐ個あるものとする。

$$AEST(n_i,J)=\max_{1 \leqq k \leqq p}\{AEST(n_{ik},PE(n_{ik}))+w(n_{ik})+r(PE(n_{ik}),J)c_{ki}\}\cdots(1)$$

【００３９】

ここで、ＰＥ（n_{ik}）は、タスクn_iのｋ番目の先行タスクを割り当てたノード番号である。$w(n_{ik})$は、先行タスクn_{ik}の処理時間を表す。ｒ(ＰＥ(n_{ik})，Ｊ)は、先行タスクの割り当てノードＰＥ（n_{ik}）と対象タスクの割り当てノードＪが同じノードであれば０、違う場合は１となる係数である。c_{ki}は、先行タスクから対象タスクへの通信時間である。式（１）において、左辺は「対象タスクn_iのノードＪにおけるＡＥＳＴ」を表し、右辺の第１項は「先行タスクn_{ik}のＡＥＳＴ」を、第２項は「先行タスクn_{ik}の処理時間」を、第３項は「先行タスクn_{ik}と対象タスクn_i間の通信時間」を表す。

【００４０】

但し、先頭のタスク（エントリタスク）については、先行タスクがひとつもないのでＡＥＳＴ＝０とする。

【００４１】

以下、図７（ｂ）に示すタスク間の依存関係を例にして、タスク４についてのＡＥＳＴを計算する手順について具体的に述べる。図７（ｂ）では、タスク４の先行タスクはタスク１～タスク３の三つあるので、ｋ＝１～３である。また、式（１）においてｉ＝４として、n_4がタスク４、n_{41}がタスク１、n_{42}がタスク２、n_{43}がタスク３となる。

【００４２】

タスク１、タスク２、タスク３がそれぞれノード１、２、３に割り当てられているとすると、ＰＥ(n_{41})＝ＰＥ(１)＝１、ＰＥ(n_{42})＝ＰＥ(２)＝２、ＰＥ(n_{43})＝ＰＥ(３)＝３となる。この条件の下、タスク４をノード２に割り当てたときのＡＥＳＴ（４，２）を式（１）にしたがって計算すると、次式のようになる（図７（ａ）のブロック１６８）。

$$AEST(4_,,2)=\max_{1 \leqq k \leqq 3}\{AEST(n_{4k},PE(n_{4k}))+w(n_{4k})+r(PE(n_{4k}),2)c_{k4}\}\cdots$$

（２）

【００４３】

式（２）においてｋ＝１としたとき、n_{41}がタスク１（ブロック１６２）に対応するから、次式が成り立つ。

$$AEST(1,PE(1))+w(1)+r(PE(1),2)c_{14}$$
$$=AEST(1,1)+w(1)+r(1,2)c_{14} \cdots (3)$$

【００４４】

式（２）においてｋ＝２としたとき、n_{42}がタスク２（ブロック１６４）に対応するから、次式が成り立つ。

$$AEST(2,PE(2))+w(2)+r(PE(2),2)c_{24}$$
$$=AEST(2,2)+w(2)+r(2,2)c_{24}$$
$$=AEST(2,2)+w(2) \cdots (4)$$

【００４５】

式（２）においてｋ＝３としたとき、n_{43}がタスク３（ブロック１６６）に対応するから、次式が成り立つ。

$$AEST(3,PE(3))+w(3)+r(PE(3),2)c_{34}$$
$$=AEST(3,3)+w(3)+r(3,2)c_{34} \cdots (5)$$

【００４６】

式（３）ないし式（５）の計算結果のうち最大のものをＡＥＳＴ（４，２）と決定する。

【００４７】

図８は、上記処理をフローチャートとして表す。まず、対象タスクn_iをあるノードＪに仮に割り当てる（Ｓ５０）。開始時刻算出部１４４は、対象タスクn_iの先行タスクのＡＥＳＴを計算し（Ｓ５２）、先行タスクの処理時間をタスク情報格納部１２０から取得する（Ｓ５４）。また、開始時刻算出部１４４は、タスク情報格納部１２０からタスク間転送データ量を、ノード情報格納部１２６からレイテンシ、スループットを取得し、対象タスクn_iをノードＪに割り当てたときの、先行タスクから対象タスクn_iへのタスク間通信時間を算出する（Ｓ５６）。先行タスクと対象タスクn_iとが同一ノードにある場合は、通信時間は「０」になる。

【００４８】

続いて、式（１）にしたがって、Ｓ５２～Ｓ５６で取得した値を加算する（Ｓ５８）。すなわち、Ｓ５２で計算した先行タスクＡＥＳＴ、Ｓ５４で取得した先行タスク処理時間、およびＳ５６で取得したタスク間通信時間を加算する。以上のＳ５０～Ｓ５８の演算を、対象タスクn_iの全ての先行タスクn_{ik}について実行し、Ｓ５８で算出される計算結果のうち最大値を、対象タスクn_iのＡＥＳＴに決定する（Ｓ６０）。

【００４９】

図９（ａ）は、ＡＬＳＴの算出方法を模式的に示しており、図９（ｂ）は、タスク間の依存関係の一例を示す。図７（ｂ）と同様、白丸内の数字がタスク番号を表しており、矢印が依存関係を表す。すなわち、図９（ｂ）では、「タスク１」の後続タスクとして「タスク４」があり、「タスク２」の後続タスクとして「タスク４」と「タスク５」があり、「タスク３」の後続タスクとして「タスク５」があることを示している。

ＡＬＳＴは、全タスクを終了させるためにそのタスクの処理を開始すべき最後のタイミングを表す。任意のノードＪにおけるあるタスクn_iのＡＬＳＴは、次式で算出される。なお、n_iはＡＬＳＴの計算対象となるタスクであり、n_{im}はタスクn_iのｍ番目の後続タスクであり、$1 \leqq m \leqq q$、すなわちn_iの後続タスクがｑ個あるとする。

$$ALST(n_i,J)=\min_{1 \leqq m \leqq q}\{ALST(n_{im},PE(n_{im}))-r(PE(n_{im}),J)c_{im}-w(n_i),\ Deadline(n_i)-w(n_i)\} \cdots (6)$$

【００５０】

ここで、ＰＥ（n_{im}）は、タスクn_iのｍ番目の後続タスクを割り当てたノード番号である。ｗ(n_i)は、対象タスクの処理時間を表す。ｒ(ＰＥ(n_{im})，Ｊ)は、後続タス

クｎ$_{im}$の割り当てノードＰＥ(ｎ$_{im}$)と対象タスクの割り当てノードＪが同じノードであれば０、違う場合は１となる係数である。ｃ$_{im}$は、対象タスクから後続タスクへの通信時間である。式（６）において、左辺は「対象タスクｎ$_{i}$のノードＪにおけるＡＬＳＴ」を表し、右辺の第１項は「後続タスクｎ$_{im}$のＡＬＳＴ」を、第２項は「対象タスクｎ$_{i}$と後続タスクｎ$_{im}$間の通信時間」を、第３項は「対象タスクｎ$_{i}$の処理時間」を表す。

【００５１】

但し、最後のタスク（出口タスク）については以下のようにする。

デッドライン時刻有りの場合：ＡＬＳＴ＝（デッドライン時間Deadline(n_i)）－（処理時間w(n_i)）

デッドライン時刻の指定なしの場合：ＡＬＳＴ＝∞

【００５２】

したがって、デッドライン時刻の指定のない経路とデッドライン時刻のある経路についてそれぞれＡＬＳＴを計算した場合は、常にデッドライン時刻有りの経路のＡＬＳＴが採用されることになる。

なお、デッドライン制約は、出口タスクだけでなく、経路の途中のタスクに設定することも可能である。

【００５３】

以下、図９（ｂ）に示すタスク間の依存関係を例にして、タスク２についてのＡＬＳＴを計算する手順について具体的に述べる。図９（ｂ）では、タスク２の後続タスクはタスク４とタスク５の二つあるので、ｍ＝１，２である。また、式（６）においてｉ＝２として、ｎ$_{2}$がタスク２、ｎ$_{21}$がタスク４、ｎ$_{22}$がタスク５となる。

【００５４】

タスク４、タスク５がそれぞれノード１、３に割り当てられているとすると、ＰＥ(ｎ$_{21}$)＝ＰＥ(４)＝１、ＰＥ(ｎ$_{22}$)＝ＰＥ(５)＝３となる。この条件の下、タスク２をノード２に割り当てたときのＡＬＳＴ（２，２）を式（６）にしたがって計算すると、次式のようになる。

$$ALST(2,2)=\min_{1\leqq m\leqq 2}\{ALST(n_{2m},PE(n_{2m}))-r(PE(n_{2m}),J)c_{2m}-w(n_2),\ Deadline(n_2)-w(n_2)\}\cdots(7)$$

【００５５】

式（７）においてｍ＝１としたとき、ｎ$_{21}$がタスク４（ブロック１７８）に対応するから、次式が成り立つ。

$$\{ALST(4,PE(4))-r(PE(4),2)c_{24}-w(2),\ Deadline(2)-w(2)\}$$
$$=\{ALST(4,1))-r(1,\ 2)c_{24}-w(2),\ Deadline(2)-w(2)\}\cdots(8)$$

【００５６】

式（７）においてｍ＝２としたとき、ｎ$_{22}$がタスク５（ブロック１８４）に対応するから、次式が成り立つ。

$$\{ALST(5,PE(5))-r(PE(5),2)c_{25}-w(2),\ Deadline(2)-w(2)\}$$
$$=\{ALST(5,3))-r(3,\ 2)c_{25}-w(2),\ Deadline(2)-w(2)\}\cdots(9)$$

【００５７】

式（８）、式（９）の計算結果のうち最小のものをＡＬＳＴ（２，２）と決定する。図９（ａ）の例では、式（８）の結果に対応するブロック１７４のＡＬＳＴ（２，２）の方が、式（９）の結果に対応するブロック１８０のＡＬＳＴ（２，２）よりも小さい、すなわち時刻が早いので、ブロック１７４のＡＬＳＴ（２，２）が採用される。

【００５８】

図１０は、上記処理をフローチャートとして表す。まず、対象タスクｎ$_{i}$をあるノードＪに仮に割り当てる（Ｓ７０）。開始時刻算出部１４４は、対象タスクｎ$_{i}$の後続タスクのＡＬＳＴを計算し（Ｓ７２）、対象タスクｎ$_{i}$の処理時間をタスク情報格納部１２０から取得する（Ｓ７４）。また、開始時刻算出部１４４は、タスク情報格納部１２０からタスク間転送データ量を、ノード情報格納部１２６からレイテンシとスループットとを取得

し、対象タスクｎ$_i$をノードＪに割り当てたときの、対象タスクｎ$_i$から後続タスクへの通信時間を算出する（Ｓ７６）。ＡＥＳＴの場合と同様に、対象タスクｎ$_i$と後続タスクとが同一ノードにある場合は、通信時間は「０」になる。

【００５９】

続いて、Ｓ７２～Ｓ７６で計算した値について、「後続タスクＡＬＳＴ－（対象タスク処理時間＋通信時間）」を計算する（Ｓ７８）。さらに、開始時刻算出部１４４は、デッドライン時刻から対象タスクの処理時間を減じたものを計算する（Ｓ８０）。Ｓ７０～Ｓ８０の一連の演算を、対象タスクｎ$_i$の先行タスクｎ$_{im}$の全てについて実行し、Ｓ７８、Ｓ８０で算出される計算結果のうち最小値を、対象タスクｎ$_i$のＡＬＳＴに決定する（Ｓ８２）。

【００６０】

続いて、図６のフローチャートの各ステップを詳述していく。

【００６１】

図１１は、図６中のタスク割り当て処Ｓ１８、Ｓ２０のフローチャートである。Ｓ１８、Ｓ２０は、割り当て対象となるタスクが異なるだけであり、同一の処理内容を繰り返す。ここでは、ノード事前指定のないタスクを別々の仮想ノードに割り当てた状態を初期状態として、各ノードを実在のノードに割り当てていく処理を実行する。タスク割り当て処理は、以下に述べるＳ３０～Ｓ４４の対象タスク選択処理と、Ｓ４６のノード選択処理を含む。

【００６２】

まず、開始時刻算出部１４４は、対象となる全タスクのＡＥＳＴを図８のフローチャートにしたがって計算し（Ｓ３０）、次に全タスクのＡＬＳＴを図１０のフローチャートにしたがって計算する（Ｓ３２）。対象タスク選択部１４６は、対象となる全てのタスクが仮想ノードでないいずれかのノードに割り当て済みか否かを判定する（Ｓ３４）。割り当て済みであれば（Ｓ３４のＹ）、タスク割り当て処理を終了する。未だ仮想ノードに割り当てられているタスクがあれば（Ｓ３４のＮ）、対象タスク選択部１４６は、ＡＬＳＴとＡＥＳＴの差分（ＡＬＳＴ－ＡＥＳＴ）を未割り当てのタスクについて計算し（以下、これを「タスク可動範囲」とも呼ぶ）、タスク可動範囲が最小となるタスクを選択する（Ｓ３６）。Ｓ３６でひとつのタスクが選択できれば（Ｓ３８のＹ）、Ｓ４０～Ｓ４４をスキップする。Ｓ３６でタスク可動範囲が等しくなるタスクが複数ある場合は（Ｓ３８のＮ）、対象タスク選択部１４６は、それらのうち、先行タスクまたは後続タスクとの間で最も長いタスク間通信時間となる経路を持つタスクを選択する（Ｓ４０）。Ｓ４０でひとつのタスクが選択できれば（Ｓ４２のＹ）、Ｓ４４をスキップする。Ｓ４２で、通信時間が等しくなるタスクが複数ある場合（Ｓ４２のＮ）、対象タスク選択部１４６は、それらのタスクのうち最小のＡＥＳＴを持つタスクを選択する（Ｓ４４）。こうして、割り当ての対象となるタスクが決定すると、ノード選択部１４８がノード選択処理を実行して割り当て対象タスクをいずれかのノードに振り分ける（Ｓ４６）。この結果、他のタスクのＡＥＳＴおよびＡＬＳＴも変化するので、Ｓ３０からの処理を繰り返す。

【００６３】

なお、Ｓ４０において、通信経路の両側のタスク、つまり送信側のタスクと受信側のタスクがともに未割り当ての段階では、タスク間通信時間を求めることができない。この場合、対象タスク選択部１４６は、受信側のタスク（換言すれば、より後続側のタスク）を優先して割り当て対象のタスクとして選択するようにしてもよい。これによって、後述するタスクのグループ化を優先して実施することができる。

【００６４】

図１１の割り当て処理は、仮想ノード（後述するグループ化のための仮想ノードを含む）に割り当てられた全てのタスクが、仮想でない実在のノードに割り当てられた時点で終了する（図１７参照）。

なお、仮想ノードのタスク間通信時間を計算する際に用いるレイテンシ、スループットについては、予め定めた固定値を使用したり、実在するノードのレイテンシ、スループッ

トの平均値を用いたりする方法が考えられる。

【００６５】

図１１のフローチャートにおける処理をまとめると、本実施の形態において、「最も重要な」タスクとは、以下の三つの基準で選択される。

評価基準１．ＡＬＳＴとＡＥＳＴの差分（タスク可動範囲）が最小のタスク

評価基準２．候補となるタスクが複数ある場合、最も長いタスク間通信時間となる経路を持つタスク

評価基準３．基準２においても候補となるタスクが複数ある場合、ＡＥＳＴが最小となるタスク

【００６６】

図１２は、ノード選択部１４８の詳細な機能ブロック図である。この図についても、これらの機能ブロックはハードウェア、ソフトウェアの組み合わせによって様々な形で実現できる。

【００６７】

ノード選択部１４８は、前処理部２００、ノード選択判定部２１０、および後処理部２３０を含む。

前処理部２００は、タスクの割当先ノードを選択するに当たり必要な処理を実行する。対象タスク確認部２０２は、割り当て対象タスクのデッドライン時刻の有無、処理時間、通信時間、ノード事前指定およびグループ事前指定があるか否かの情報を、格納部１３０から取得する。先後タスク確認部２０４は、割り当て対象タスクの先行タスクおよび後続タスクについての情報を格納部１３０から取得する。ノードリスト作成部２０６は、対象タスク確認部２０２および先後タスク確認部２０４の情報を参考にして、割り当て対象タスクを配置可能なノード情報が含まれるノードリストを作成する。

【００６８】

ノード選択判定部２１０は、ＡＥＳＴ、ＡＬＳＴおよび、前処理部２００で準備された情報などに基づいて、割り当て対象タスクを割り当てるノードを選択する。

ノードリスト作成部２０６で作成されたノードリストは、ノードリスト格納部２２０に格納される。また、図５の開始時刻算出部１４４で計算されたＡＥＳＴおよびＡＬＳＴは、開始時刻格納部２２２に記憶されている。

【００６９】

空き時間検出部２１４は、割り当て対象タスクを仮に割り当てるノード（以下、「候補ノード」という）を選択した後、開始時刻格納部２２２内のＡＥＳＴおよびＡＬＳＴを使用して、候補ノード上での割り当て対象タスクの実行が可能となる空き時間を検出する。より具体的には、各ノードに割り当て対象タスクを割り当てた場合の仮ＡＥＳＴを計算する。割り当て対象タスクを候補ノードに割り当て可能であれば、その仮ＡＥＳＴに開始するように割り当てたと仮定し、割り当て対象タスクの最も重要な後続タスクを、最も早く開始できる時刻（後続タスクＡＥＳＴ）を求める。そして、後続タスクＡＥＳＴが最小となるノードを割り当て対象タスクの割当先として選択する。候補ノードが複数ある場合には、仮ＡＥＳＴが最小のものを優先する。ＡＥＳＴ条件判定部２２４は、空き時間検出部２１４で算出されるＡＥＳＴの条件判定を実行する。

なお、「最も重要な後続タスク」は、割り当て対象タスクの後続タスクの中から、上述の評価基準１～３にしたがって決定される。このとき、ＡＬＳＴとＡＥＳＴの差分、およびＡＥＳＴについては、各後続タスクについての値を用いるが、タスク間通信時間については、割り当て対象タスクと各後続タスクとの間の通信時間を用いることに注意する。

【００７０】

後処理部２３０は、ノード選択判定部２１０で選択されたノードを受け取り、必要な後処理を実行する。後処理部２３０は、必要に応じてタスクのグループ化を実行するグループ化部２２６を含む。

【００７１】

図１３は、ノード選択処理のフローチャートである。前処理では、割り当て対象タスク

n_iの割当先候補となるノードを列挙するノードリストを作成する（Ｓ９０）。ノードリスト内の各ノードに対してメインループを実行して、割り当て対象タスクn_iを配置するノードを選択する（Ｓ９２）。後処理では、割り当て対象タスクn_iを選択されたノードに割り当てた上で、そのノードが後述する仮想ノードの場合には、タスクのグループ化を実行する（Ｓ９４）。

【００７２】

図１４は、図１３におけるＳ９０の前処理のフローチャートである。

ノードリスト作成部２０６は、割り当て対象タスクn_iのノード事前指定があるか否かを判定する（Ｓ１００）。ノード事前指定があれば（Ｓ１００のＹ）、そのノードをノードリストに加える（Ｓ１０２）。ノード事前指定がなければ（Ｓ１００のＮ）、割り当て対象タスクn_iを配置可能な空きリソースを有するノードを検索し、そのノードをノードリストに加える（Ｓ１０４）。

【００７３】

次に、ノードリスト作成部２０６は、割り当て対象タスクn_iにデッドライン時刻が存在するか否かを確認する（Ｓ１０６）。デッドライン時刻がない場合（Ｓ１０６のＮ）、続くＳ１０８、Ｓ１１０をスキップする。デッドライン時刻がある場合（Ｓ１０６のＹ）、さらに、割り当て対象タスクの最も重要な先行タスクがいずれかのノードに割り当て済みか否かを判定する（Ｓ１０８）。割り当て済みであれば（Ｓ１０８のＮ）、Ｓ１１０をスキップし、割り当て済みでなければ（Ｓ１０８のＹ）、ノードリストに仮想ノードを追加する（Ｓ１１０）。この仮想ノードは、タスクを一時的にグループ化するために使用する。そして、メインループで使用する各変数に初期値をセットする（Ｓ１１２）。これで前処理は終了する。

なお、「最も重要な先行タスク」は、割り当て対象タスクの先行タスクの中から、上述の評価基準１～３にしたがって決定される。このとき、ＡＬＳＴとＡＥＳＴの差分、およびＡＥＳＴについては、各先行タスクについての値を用いるが、タスク間通信時間については、各先行タスクと割り当て対象タスクとの間の通信時間を用いることに注意する。

【００７４】

図１５および図１６は、図１３におけるＳ９２のメインループのフローチャートである。

空き時間検出部２１４は、ノードリスト内の全ノードについての検出処理が終了したか否かを判定する（Ｓ１２０）。終了していない場合は、ノードリストからひとつのノードを候補ノードＪとして選択し（Ｓ１２２）、候補ノードＪに割り当て対象タスクn_iを割り当てるだけのリソースが余っているか否かを判定する（Ｓ１２３）。リソースに余裕がなければ（Ｓ１２３のＮ）、Ｓ１２０に戻る。リソースが余っていれば（Ｓ１２３のＹ）、そのノードについて空き時間検出処理を実行して、割り当て対象タスクn_iの仮ＡＥＳＴを求める（Ｓ２１４）。仮ＡＥＳＴが算出可能であれば（Ｓ１２６のＹ）、図１６に進む。仮ＡＥＳＴが算出不可能であれば（Ｓ１２６のＮ）、Ｓ１２０に戻る。

【００７５】

図１６に移り、空き時間検出部２１４は、対象タスクn_iの重要後続タスクn_cがあるか否かを判定する（Ｓ１４０）。重要後続タスクn_cがある場合（Ｓ１４０のＹ）、タスクn_iを候補ノードＪに仮に割り当てる（Ｓ１４２）。空き時間検出部２１４は、タスクn_cのノード指定があるか否かを判定する（Ｓ１４６）。このノード指定は、ノード事前指定と、このアルゴリズムにより割当先が決定された場合の両方を含む。ノード指定がない場合は（Ｓ１４６のＮ）、空き時間検出処理により、後続タスクn_cを候補ノードＪに割り当てたときの後続タスクのＡＥＳＴを計算する（Ｓ１４８）。ノード指定がある場合には（Ｓ１４６のＹ）、指定ノードに後続タスクn_cを割り当てたときの後続タスクのＡＥＳＴを計算する（Ｓ１５０）。重要後続タスクn_cが既に割り当て済みであれば、対象タスクn_iを割り当てたことで値が変わるので、重要後続タスクn_cのＡＥＳＴを再計算する。重要後続タスクn_cにノード事前指定があり、かつその指定されたノードに十分なリソースがない場合、重要後続タスクn_cのＡＥＳＴを「∞」とする。なお、重要後続タ

スクn_cが存在しない場合には、後続タスクＡＥＳＴを「０」とする。後続タスクのＡＥＳＴを計算すると、仮に割り当てた割り当て対象タスクn_iを元に戻す（Ｓ１５２）。

【００７６】

次に、ＡＥＳＴ条件判定部２２４は、Ｓ１４８またはＳ１５０で求めた後続タスクＡＥＳＴが、Ｓ１５６でセットされる最小後続タスクＡＥＳＴ未満か否かを判定する（Ｓ１５４）。つまり、割り当て対象タスクn_iの割当先ノードには、後続タスクＡＥＳＴが最小のものが優先される。これは、後続タスクのＡＥＳＴが小さいほど、ノード経路を短くしてタスク間通信時間を短縮できるためである。後続タスクＡＥＳＴが最小であれば（Ｓ１５４のＹ）、ＡＥＳＴ条件判定部２２４は、候補ノードＪを暫定的にベストノードにセットし、最小後続タスクＡＥＳＴを現在の後続タスクＡＥＳＴで書き換える。さらに、最小対象タスクＡＥＳＴを、対象タスク仮ＡＥＳＴで書き換える（Ｓ１５８）。フローは図１５のＳ１２０に戻り、ノードリスト内の別のノードについて上記処理を繰り返す。

【００７７】

後続タスクＡＥＳＴが最小後続タスクＡＥＳＴ以上の場合は（Ｓ１５４のＮ）、ＡＥＳＴ条件判定部２２４は、さらに、後続タスクＡＥＳＴが最小後続タスクＡＥＳＴと等しく、かつ、仮ＡＥＳＴがＳ１５８でセットされる最小仮ＡＥＳＴ未満であるかを判定する（Ｓ１５６）。後続タスクＡＥＳＴが最小後続タスクＡＥＳＴと等しく、かつ、仮ＡＥＳＴが最小であれば（Ｓ１５６のＹ）、Ｓ１５８に進む。後続タスクＡＥＳＴが最小後続タスクＡＥＳＴより大きいか、または仮ＡＥＳＴが最小でない場合、対象タスクをこの候補ノードＪに割り当てるべきでないので、フローはＳ１２０に戻り、別のノードについて上記処理を繰り返す。

【００７８】

図１７は、図１３におけるＳ９４の後処理のフローチャートである。

図１５のＳ１２０において、ノードリスト内の全ノードについての処理が終了すると、フローは図１７に移る。後処理部２３２は、上記処理でベストノードが存在するか否かを判定する（Ｓ１７０）。ベストノードが存在しない場合、つまり割り当て対象タスクn_iの割当先ノードが見つからなかった場合には（Ｓ１７０のＮ）、このノード選択処理は失敗であり（Ｓ１７２）、適切なフォロー処理を実行する。ベストノードが存在する場合（Ｓ１７０のＹ）、割り当て対象タスクn_iをそのベストノードに割り当てる（Ｓ１７４）。割り当てられたノードが仮想ノードである場合（Ｓ１７６のＹ）、グループ化部２２６がタスクのグループ化処理を実行した上で（Ｓ１７８）、ノードリストをリセットし（Ｓ１８０）、今回のノード選択処理は成功となる（Ｓ１８２）。割り当てられたノードが仮想ノードでない場合（Ｓ１７６のＮ）、Ｓ１７８およびＳ１８０はスキップする。

【００７９】

図１８は、図１５のＳ１２４、図１６のＳ１４８およびＳ１５０の空き時間検出処理の詳細なフローチャートである。空き時間検出部２１４は、候補ノードＪに割り当て済みの全てのタスクについて、ＡＥＳＴとＡＬＳＴを計算する（Ｓ１９０）。なお、この計算は開始時刻算出部１４４によって実行されてもよい。次に、対象タスクを配置可能な位置を選択し（Ｓ１９２）、先行タスクの終了時刻から後続タスクの開始時刻までの間に、対象タスクn_iの処理時間分の空きがあるか否かを判定する（Ｓ１９４）。空き時間があれば（Ｓ１９４のＹ）、そのタスクのＡＥＳＴを出力する（Ｓ１９６）。空き時間がなければ（Ｓ１９４のＮ）、そのノード内で、対象タスクを配置する可能性のある全ての位置について検討したか否かを判定し（Ｓ１９８）、未検討の配置があれば（Ｓ１９８のＮ）、Ｓ１９２に戻り他の配置について検討する。全ての位置について確認した場合には（Ｓ１９８のＹ）、「ＰＵＳＨ型挿入」による配置が可能か否かを判定する（Ｓ２００）。

【００８０】

ここで、ＰＵＳＨ型挿入について説明する。既に候補ノードＪに割り当て済みの各タスクのＡＬＳＴを遅延させることによって、対象タスクn_i（グループ化されたタスクの場合は、同じグループのタスク全て）を候補ノードＪに割り当てられるか否かを検討する。つまり、アプリケーション全体の終了時刻が遅れることを許容して、対象タスクを必ずい

ずれかの実在のノードに割り当てるようにする。

ＰＵＳＨ型挿入による配置が可能であれば（Ｓ２００のＹ）、そのタスクのＡＥＳＴを出力する（Ｓ１９６）。ＰＵＳＨ型挿入による配置が不可能であれば（Ｓ２００のＮ）、このノードＪには対象タスクの割り当てが不可能となる。

【００８１】

図１９（ａ）、図１９（ｂ）は、図１８の空き時間検出処理を模式的に説明する図である。空き時間検出処理では、既に候補ノードＪに割り当て済みの各タスクの最遅開始時刻ＡＬＳＴを遅らせずに、対象タスクn_iを候補ノードＪに割り当てることができるか否かを判定する。

空き時間検出部２１４は、候補ノードＪにタスクn_{jk}とタスクn_{jk+1}が既に割り当て済みであるとき、割り当て対象タスクn_iをさらに配置可能か否かを判定する。但し、対象タスクn_iの挿入位置（つまり、タスクn_{jk}とタスクn_{jk+1}の間）より後には、対象タスクn_iの先行タスクが存在せず、挿入位置より前には、対象タスクn_iの後続タスクが存在しないようにする。

【００８２】

タスクn_jの処理終了時刻は、ＡＥＳＴとタスクn_jの処理時間を用いて、$\{ALST(n_j,J)+w(n_j)\}$で表せる。また、タスクn_jの最先開始時刻は、$AEST(n_j,J)$で表せる。したがって、割り当て対象タスクをタスクn_{jk}とタスクn_{jk+1}の間に配置できるかは、タスクn_{jk}の終了時刻とタスクn_{jk+1}の開始時刻の差分である「タスク実行可能範囲」が、対象タスクn_iの処理時間以上であればよいので、次式が成立すればよい。

$$\min\{ALST(n_i,J)+w(n_i),\ ALST(n_{jk+1},J)\}-\max\{AEST(n_i,J),\ AEST(n_{jk},J)+w(n_{jk})\}-(AEST(n_i)-ALST(n_i))\geqq w(n_i)\quad\cdots\quad(10)$$

【００８３】

第１項では、タスクn_{jk+1}の最遅開始時刻と、対象タスクn_iの処理終了時刻のうち最も遅い場合とを比較して、時刻が早い方が選択される。第２項では、対象タスクn_iの最先開始時刻と、タスクn_{jk}の処理終了時刻のうち最も早い場合とを比較して、時刻が遅い方が選択される。これらの差分が対象タスクn_i自体の処理時間よりも長ければ、対象タスクn_iをタスクn_{jk}とタスクn_{jk+1}との間に配置することが可能であり、差分が対象タスクn_i自体の処理時間よりも短ければ、対象タスクn_iをそれらの間に配置することが不可能であると判定される。なお、第３項は、ＡＥＳＴとＡＬＳＴの基準となる時刻が異なることに基づく補正項である。

【００８４】

割り当て可能であれば、空き時間検出部２１４は、対象タスクn_jを配置できる最先のＡＥＳＴを返す。割り当て不可であれば、上述の「ＰＵＳＨ型挿入」による配置を検討する。

【００８５】

図１９（ａ）、図１９（ｂ）を参照して、さらに説明する。図１９（ａ）のケースでは、対象タスクn_iの最先開始時刻$AEST(n_i,J)$の方がタスクn_{jk}の処理終了時刻$AEST(n_{jk},J)+w(n_{jk})$よりも遅い時刻であるため、対象タスクn_iの最先開始時刻が採用される。また、対象タスクn_iの処理終了時刻$ALST(n_i,J)+w(n_i)$は、タスクn_{jk+1}の最遅開始時刻$ALST(n_{jk+1},J)$よりも早い時刻であるため、対象タスクn_iの処理終了時刻が採用される。したがって、このケースでは対象タスクn_iの実行可能範囲は次式のようになる。

$$(\text{実行可能範囲})=\{ALST(n_i,J)+w(n_i)\}-AEST(n_i,J)\quad\cdots\quad(11)$$

【００８６】

図１９（ｂ）のケースでは、タスクn_{jk}の処理終了時刻$AEST(n_{jk},J)+w(n_{jk})$の方が対象タスクn_iの最先開始時刻$AEST(n_i,J)$よりも遅い時刻であるため、タスクn_{jk}の処理終了時刻が採用される。また、タスクn_{jk+1}の最遅開始時刻$ALST(n_{jk+1},J)$の方が対象タスクn_iの処理終了時刻$ALST(n_i,J)+w(n_i)$よりも早い時刻であるため、タスクn_{jk+1}のＡＬＳＴが採用される。したがって、このケースでは対象タスクn_iの実行可能

範囲は次式のようになる。

（実行可能範囲）＝$ALST(n_{jk+1},J)-\{AEST(n_{jk},J)+w(n_{jk})\}$・・・（１２）

【００８７】

図２０は、図１８のＳ２００における「ＰＵＳＨ型挿入」を説明する図である。図２０の左側のブロック２７０に示すように、対象タスクn_iの実行可能範囲内では、対象タスクn_iの処理時間をタスクn_{jk+1}の処理時間と重ねない限り対象タスクn_iをノードＪに配置できない場合がある。このようなときは、空き時間検出部２１４は、図２０の右側のブロック２７２に示すように、タスクn_{jk+1}のＡＬＳＴを変更してタスクn_{jk+1}の開始時刻を後ろにずらすことによって、対象タスクn_iの配置を実現する。これは、アプリケーション全体の終了時刻を遅らせることにつながる。

【００８８】

以上説明したように、空き時間検出処理では、タスクの割当先ノードの選択方法として、対象タスクとその最も重要な後続タスク（すなわち、未だ仮想ノードに割り当てられている後続タスクのうち、そのＡＥＳＴとＡＬＳＴとの差が最小のタスク）とを割り当てた際に、後続タスクＡＥＳＴが最小となるノードを選ぶようにした。これによって、対象タスクの１ステップ後の後続タスクの割り当てまでを考慮して対象タスクの割当先ノードが選択されることになるので、対象タスクのＡＥＳＴは早期にあるが、後続タスクのＡＥＳＴが遅れ、結果として全体の処理時間が長期化してしまうような事態を回避することができる。

【００８９】

（実施例）

本発明の実施の形態について詳細に説明した。以降では、実施の形態において述べた各処理を適用してタスクをノードに割り当てる様子を、具体例を参照しつつ説明する。

【００９０】

図１に示した５つのノードを含む分散アプリケーション実行システムにおいて、図２１に示す先後関係のあるタスクを処理する場合について説明する。図中、白丸内の数字は、「プレイヤーの番号－タスク番号」を表しており、例えば「１－１」は、「プレイヤー１の１番目のタスク」を表している。図２１では、プレイヤー１に関連したタスクとして「１－１」～「１－６」、プレイヤー２に関連したタスクとして「２－１」～「２－６」が含まれ、さらに、上述した対戦ゲームアプリケーションの衝突処理のように、両プレイヤーに対しての演算を同一ノードで実行すべきタスクとして「３－１」が規定される。

【００９１】

さらに、ノード事前指定として、タスク「１－１」「１－６」についてはノード１が指定され、タスク「２－１」「２－６」についてはノード５が指定されているとする。これらは、コントローラによる入力処理やディスプレイへの画面表示などの、各プレイヤーのノードでのみ実施されるべきタスクである。また、タスク「３－１」についてはノード４が指定されているとする。

【００９２】

また、デッドライン時刻として、プレイヤー１の経路については２００ｍｓ、プレイヤー２の経路については２５０ｍｓが設定されている。

さらに、説明を簡単にすべく、各タスク間の通信時間計算のためのレイテンシは５ｍｓ、スループットは１００Ｍｂｐｓに統一する。また、各ノードとも計算リソースは十分に空いているものとする。

【００９３】

まず、情報収集部１０８により、各タスクの処理時間、レイテンシ、スループット、転送データ量を取得する。図２２は、それらを取得した後のタスク相関図であり、図中にそれらの値が記載してある。白丸の左下または右下の数字がタスクの処理時間であり、矢印にそって書かれた数字が転送データ量である。タスク間通信時間は、「レイテンシ＋（スループット×転送データ量）」で算出される。

【００９４】

続いて、ノード事前指定されているタスクがそれぞれのノードに配置される。このとき、タスク間の先後関係は当然考慮され、また必要に応じて各ノードの空きリソース量も確認される。

【００９５】

続いて、ノード事前指定のないタスクを仮想ノードに配置し、開始時刻算出部１４４により、各タスクについてＡＥＳＴおよびＡＬＳＴが計算される。この計算には、先に述べたデッドライン時刻から、タスクの処理時間、通信時間を用いて、上述した数式を使用して計算される。例えば、タスク２－６については、デッドライン時刻２５０ｍｓからタスク２－６の処理時間１０ｍｓを減じた２４０ｍｓがＡＬＳＴとなる。タスク２－５については、タスク２－６のＡＥＳＴ２４０ｍｓから、５Ｍｂ分の通信時間５０ｍｓと、レイテンシ５ｍｓと、タスク２－５の処理時間２０ｍｓを減じた１６５ｍｓがＡＬＳＴとなる。他のタスクについても同様である。なお、タスク３－１については、タスク１－６から辿る左側の経路と、タスク２－６から辿る右側の経路の両方から算出するため、ＡＬＳＴが二つ求められるが、このような場合は、左側の経路から求められた、より小さい値である－５ｍｓがＡＬＳＴとなる。

【００９６】

ＡＥＳＴについては、タスク１－１および２－１のＡＥＳＴを０とし、そこからタスク処理時間と通信時間を加えていく。例えば、タスク１－２であれば、タスク１－１の処理時間１０ｍｓ、１Ｍｂ分の通信時間１０ｍｓ、およびレイテンシ５ｍｓを加えた２５ｍｓがＡＥＳＴとなる。他のタスクについても同様である。

【００９７】

次に、（ＡＬＳＴ－ＡＥＳＴ）で求められるタスク可動範囲を算出する。以上の計算の結果、各タスクのＡＥＳＴ、ＡＬＳＴ、タスク可動範囲は、図２３に示す表のようになる。

【００９８】

続いて、ノードの選択に移る。最も余裕のない経路、すなわちタスク可動範囲が最小（－１３０ｍｓ）のタスクのうち、通信時間の最も長いものを検出する。図２３から分かるように、これに適合するのはタスク１－４とタスク１－５間の経路（以下、経路Ａという）、およびタスク１－５とタスク１－６間の経路（以下、経路Ｂという）である。二つの経路Ａ、Ｂの通信時間は５５ｍｓで等しいので、対象タスクを決定するために、それらの後続タスクのＡＥＳＴについてさらに検討する。この場合、経路Ａの後続タスク１－５のＡＥＳＴ（２４５ｍｓ）と経路Ｂの後続タスク１－６のＡＥＳＴ（３２０ｍｓ）を比較すると、タスク１－５のＡＥＳＴの方が小さいので、まずタスク１－５の配置を先に考慮する。

【００９９】

代替的に、二つの経路Ａ、Ｂの通信時間が５５ｍｓで等しいとき、次のような手順で対象タスクを決定してもよい。まず、経路Ａについて考えると、タスク１－４とタスク１－５はどちらも未割り当てなので、上述したように、グループ化を優先して行うべく後続側のタスク１－５を割り当て候補とする。次に、経路Ｂについて考えると、タスク１－６は割り当て済みなので、タスク１－５が割り当て候補となる。したがって、タスク１－５の配置を先に考慮する。

【０１００】

ここで、空き時間検出処理により、タスク１－５のベストノード、後続タスクＡＥＳＴ、仮ＡＥＳＴを求める。最初の状態では、タスク１－５の先行タスクである１－４が未割り当てなので、タスク１－４だけを割り当てた仮想ノード０を想定する。この時点では、タスク１－５のベストノードはヌル値であり、後続タスクＡＥＳＴ、仮ＡＥＳＴは「∞」である。また、タスク１－５を割り当て可能なノードのリストには、仮想ノード０およびノード１～５の６つのノードが登録される。ここで、まずリストの先頭にある仮想ノード０にタスク１－５を割り当てる場合を考える。上記式（２）の計算をすると、仮想ノード０に割り当てた場合のタスク１－５の仮ＡＥＳＴは１９０ｍｓになる。

【０１０１】
次に、重要後続タスクｎ$_c$として、タスク１－６を選択する。タスク１－５を仮想ノード０に割り当てた場合は、タスク１－６のＡＥＳＴは２６５ｍｓになる。
したがって、後続タスクＡＥＳＴ（２６５ｍｓ）＜最小後続タスクＡＥＳＴ（∞）になるので、ベストノードに仮想ノード「０」、最小後続タスクＡＥＳＴに「２６５」、最小仮ＡＥＳＴに「１９０」が代入される。
【０１０２】
上記と同様の計算を、残りのノード１～５について繰り返す。Ｊ＝１のとき、後続タスクＡＥＳＴは２６５ｍｓ、仮ＡＥＳＴは２４５ｍｓになるため、ベストノードは０のまま変更されない。Ｊ＝２～５についても同様である。
このように、ＤＣＰ法とは異なり、後続タスクＡＥＳＴの小さい方を優先するが、後続タスクＡＥＳＴが同一の場合には、仮ＡＥＳＴが小さい方を優先する。
【０１０３】
その結果、ベストノードは「０」になるので、タスク１－５は仮想ノード０に割り当てられる。すなわち、タスク１－４とは同じノードに割り当てるべきと判断され、タスク１－４と１－５はグループ化される（図２４参照）。グループ化されたタスクは、以後の計算ではひとつのタスクと見なされて処理される。すなわち、グループ内のタスク間の通信時間は０であり、タスクの処理時間のみ加算されたもので計算する。
【０１０４】
１－４と１－５がグループ化された状態で、全タスクのＡＥＳＴとＡＬＳＴを更新する。グループ化されて同一ノードに配置されたため、１－４と１－５の間の通信時間が「０」になることで、全タスクのＡＥＳＴ、ＡＬＳＴが更新される。更新後の各タスクのＡＥＳＴ、ＡＬＳＴ、タスク可動範囲を図２５に示す。
【０１０５】
この結果、今度はタスク可動範囲が－８０ｍｓのタスクが割り当て対象になり、これらのうち通信時間の最も長い経路として、タスク２－４と２－５の間と、タスク２－５と２－６の間（５５ｍｓ）が検出される。上述と同様に、タスク２－５について、ノード選択処理が実行される。その結果、２－４と２－５はグループ化される。
【０１０６】
次に、タスク１－４と１－５のグループについての計算が実施され、その計算結果に従うと、タスク１－４、１－５、１－６は同一のグループになる。ここで、タスク１－６は、ノード１に配置されるよう指定されている。したがって、タスク１－４、１－５、１－６は、ノード１に割り当てられることが決定される（図２６参照）。
このように計算を繰り返していくことによって、最終的に各タスクは図２７に示すようにノードに割り当てられる。
【０１０７】
以上説明したように、実施の形態によれば、最先開始時刻ＡＥＳＴと最遅開始時刻ＡＬＳＴの差分であるタスク可動範囲が最大のタスクからノードへの割り当てを実行する。また、タスクのノードへの割り当てを実行する際に、重要な経路、すなわちタスク間の通信時間が最大になるタスクを割り当てるノードから順に割り当てを実行するようにした。これによって、通信遅延時間を効率的に削減することができる。また、より重要なタスクから順に割り当てが決定されるため、重要度の低いタスクによってリソースが先に消費されることが回避され、タスクの実施に必要なリソースを先に確保することができる。
【０１０８】
また、最遅開始時刻ＡＬＳＴを算出する際に、タスクのデッドライン時刻を考慮して算出するようにした。これによって、タスクの終了期限に対する余裕時間を考慮して、最遅開始時刻ＡＬＳＴを計算することができる。
【０１０９】
また、タスク割り当てを実行することによって、ネットワーク内の複数のノード間で計算リソース（例えば、ＣＰＵ時間、メモリ容量など）を融通することができる。したがっ

て、単一機器の性能以上の能力を発揮させることも可能となる。

【０１１０】

従来は、分散ネットワーク環境における格闘ゲームなどでは、衝突判定を専門に担当する専用サーバを準備して整合性を維持していた。この方式では、サーバとノード間の通信による遅延時間が大きくなりがちである。また、参加プレイヤー数の増加につれ、衝突判定の演算量も増大するので、サーバを増強する必要がある。

これに対し、本実施の形態では、専用サーバを用意せず全ての演算はノードで実行されるため、専用サーバの増強を考える必要はない。

【０１１１】

また、本発明の仕組みを採用することによって、並列システムや分散システムでもリアルタイム処理を実行することができる。

従来では、ネットワークに存在するノード数、各ノードの性能、ネットワークの構造などが事前に分からない場合や、アプリケーション実行中にノードの増減が発生する分散ネットワーク環境では、ノード間でタスクをリアルタイムでやりとりする処理は不可能であった。これに対し、本発明では、ノード間の接続を組み替えることによってネットワークの構造が変化したり、または、ノードの追加や減少などが生じた場合であっても、ノードについての情報を処理することによって、適切なタスク割り当て処理を継続することができる。

【０１１２】

さらに、本発明では、特定のタスクを割り当てるノードを事前に指定することが可能なので、キー入力や音声出力、画像出力などの、特定ユーザの担当ノードで実行しなければ意味のないタスクを必ず担当ノードに割り当てることができる。

また、複数のタスクで使用する同じコンテキストデータを持つタスクを、グループ化によって同じノードにまとめて割り当てるように指定することも可能である。これによって、通信量および通信回数を削減することができ、また通信遅延の影響を最小にすることができる。

このように、本発明では、ノード事前指定を活用することによって、多様な種類および性質のタスクを扱うことができる。

【０１１３】

本発明では、従来の方法と比較してタスク割り当てに要する計算量は増加する傾向にあるが、上述した様々な特徴のようにタスクを割り当てるノード選択の柔軟性が高いため、全体の処理をより早期に終了可能なタスク配置が実現されうる。

【０１１４】

以上、実施の形態をもとに本発明を説明した。これらの実施の形態は例示であり、各構成要素またはプロセスの組み合わせにいろいろな変形例が可能なこと、またそうした変形例も本発明の範囲にあることは当業者に理解されるところである。

【０１１５】

実施の形態で述べた構成要素の任意の組み合わせ、本発明の表現を方法、装置、システム、コンピュータプログラム、記録媒体などの間で変換したものもまた、本発明の態様として有効である。また、本明細書にフローチャートとして記載した方法は、その順序にそって時系列的に実行される処理のほか、並列的または個別に実行される処理をも含む。

【０１１６】

本発明の一連の処理をソフトウェアにより実行させる場合には、そのソフトウェアを構成するプログラムが専用のハードウェアに組み込まれているコンピュータとして実現してもよいし、あるいは、各種のプログラムをインストールすることで各種の機能を実行することが可能な汎用のコンピュータに、ネットワークや記録媒体からソフトウェアをインストールすることによって実現してもよい。

【０１１７】

実施の形態では、複数ノードがネットワークで接続された分散アプリケーション実行環境について述べたが、ひとつのノードに複数のプロセッサが存在するマルチプロセッサシ

ステムにおいて、それらプロセッサ間で処理を分担する「並列アプリケーション実行環境」に対しても、本発明を適用することができる。この場合、分散環境でのノード間のレイテンシ、スループットを、ノード内のプロセッサ間のレイテンシ、スループットで置き換えることで、上述と同様のアルゴリズムを適用できる。

さらに、それらマルチプロセッサシステムのノード同士がネットワークで接続された環境に対しても、本発明を適用することができる。この場合、ひとつのノード内に存在する複数のプロセッサ間のレイテンシ、スループットと、別ノードのプロセッサまでのレイテンシ、スループットとを適切に設定することによって、上述と同様のアルゴリズムを適用することができる。

【図面の簡単な説明】

【０１１８】

【図１】実施の形態に係る分散アプリケーション実行システムの構成図である。

【図２】各ノードを構成する汎用コンピュータの概略構成を示す図である。

【図３】各ノードで実行されるタスクと、タスク間の実行順序の制約を表す図である。

【図４】タスク割り当て結果の一例を示す図である。

【図５】実施の形態に係るタスク割り当て処理を実行するノードの機能ブロック図である。

【図６】タスク割り当て処理のメインフローチャートである。

【図７】図７（ａ）はＡＥＳＴの算出方法を模式的に示し、図７（ｂ）はタスク間の依存関係の一例を示す図である。

【図８】ＡＥＳＴの計算処理のフローチャートである。

【図９】図９（ａ）はＡＬＳＴの算出方法を模式的に示し、図９（ｂ）はタスク間の依存関係の一例を示す図である。

【図１０】ＡＬＳＴの計算処理のフローチャートである。

【図１１】図６におけるタスク割り当て処理のフローチャートである。

【図１２】ノード選択部の詳細な機能ブロック図である。

【図１３】図１１におけるノード選択処理のフローチャートである。

【図１４】図１３における前処理の詳細なフローチャートである。

【図１５】図１３におけるメインループの詳細なフローチャートである。

【図１６】図１３におけるメインループの詳細なフローチャートである。

【図１７】図１３における後処理の詳細なフローチャートである。

【図１８】図１５、図１６における空き時間検出処理のフローチャートである。

【図１９】図１９（ａ）、図１９（ｂ）は、空き時間検出処理の計算手法を模式的に示す図である。

【図２０】図１８のＳ２００における「ＰＵＳＨ型挿入」を説明する図である。

【図２１】タスク割り当ての実行対象となるタスク経路の具体例を示す図である。

【図２２】タスク処理時間、通信時間、事前指定ノードを記載したタスク経路を示す図である。

【図２３】図２２の各タスクのＡＥＳＴ、ＡＬＳＴ、タスク可動範囲を示す表である。

【図２４】タスク１－４とタスク１－５とをグループ化した後のタスク経路を示す図である。

【図２５】図２４の各タスクのＡＥＳＴ、ＡＬＳＴ、タスク可動範囲を示す表である。

【図２６】タスク１－４、１－５、１－６をノード１に割り当てた後のタスク経路を示す図である。

【図２７】タスク割り当て処理後のタスク経路を示す図である。

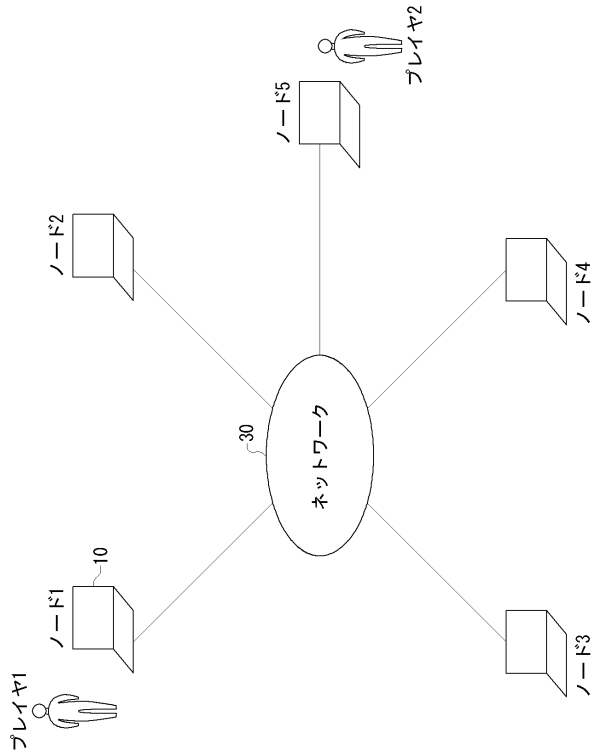
【符号の説明】

【０１１９】

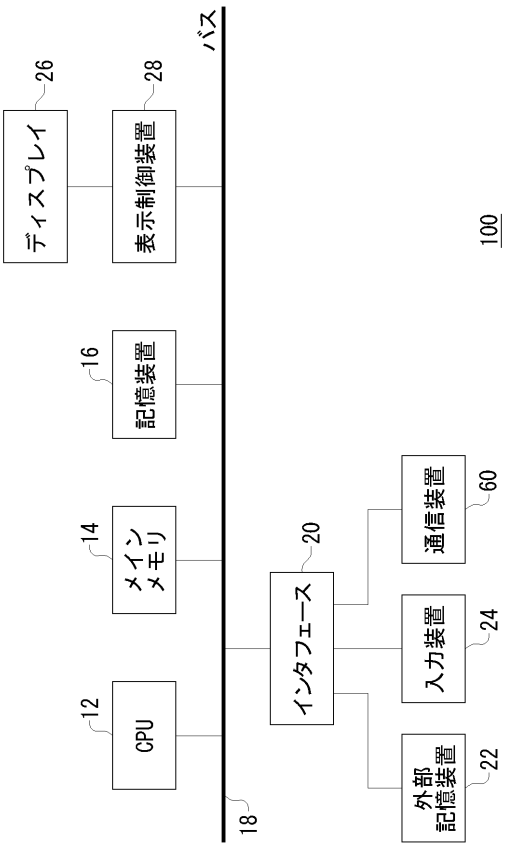
１０　ノード、　１００　コンピュータ、　１０２　ユーザ入力部、　１０４　プログラム解析部、　１０８　情報収集部、　１１０　タスク情報取得部、　１１２　ノード情報取得部、　１１８　事前指定受付部、　１２０　タスク情報格納部、　１２６　ノード

情報格納部、　１３０　格納部、　１４０　タスク割り当て部、　１４４　開始時刻算出部、　１４６　対象タスク選択部、　１４８　ノード選択部、　１５０　タスク配置部、　２００　前処理部、　２０２　対象タスク確認部、　２０４　先後タスク確認部、　２０６　ノードリスト作成部、　２１０　ノード選択判定部、　２１４　空き時間検出部、　２２０　ノードリスト格納部、　２２２　開始時刻格納部、　２２４　ＡＥＳＴ条件判定部、　２２６　グループ化部、　２３０　後処理部。

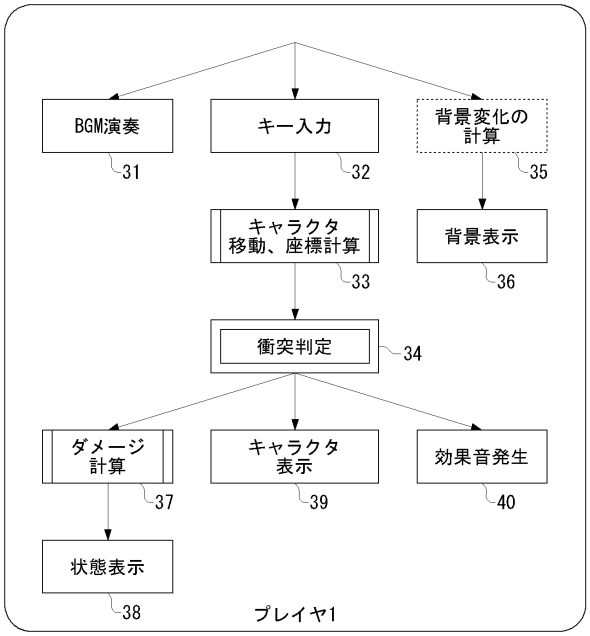
【図１】



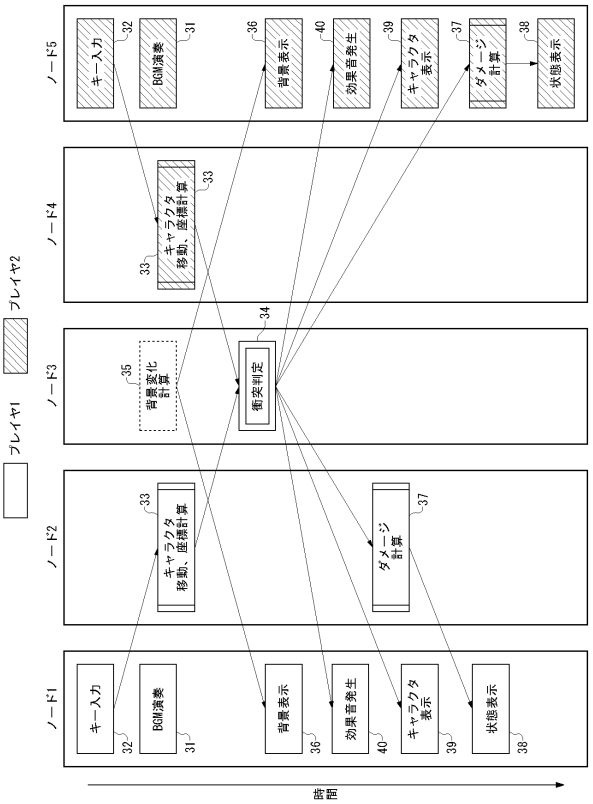
【図２】



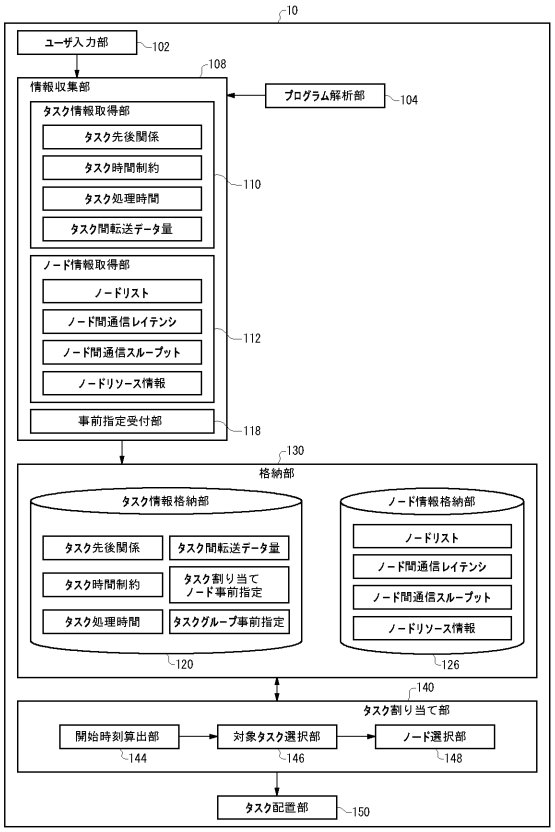
【図３】


BGM演奏
31
キー入力
32
背景変化の計算
35
キャラクタ移動、座標計算
33
背景表示
36
衝突判定
34
ダメージ計算
37
キャラクタ表示
39
効果音発生
40
状態表示
38
プレイヤ1

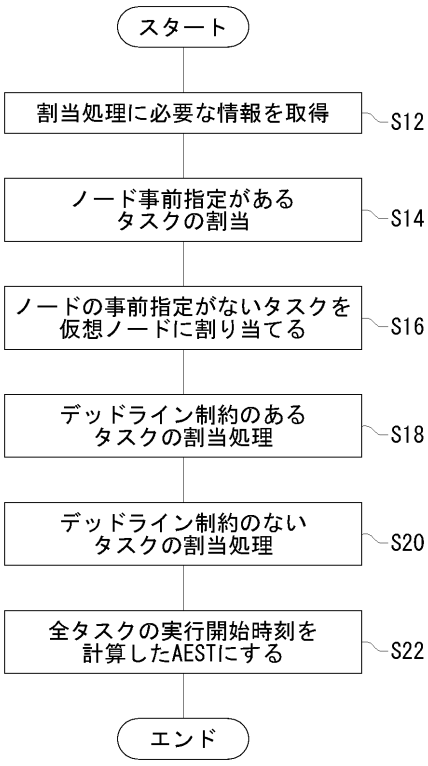
【図４】


ノード5
ノード4
ノード3
ノード2
ノード1
プレイヤ2
プレイヤ1
キー入力
BGM演奏
背景表示
効果音発生
キャラクタ表示
ダメージ計算
状態表示
キャラクタ移動、座標計算
背景変化計算
衝突判定
時間

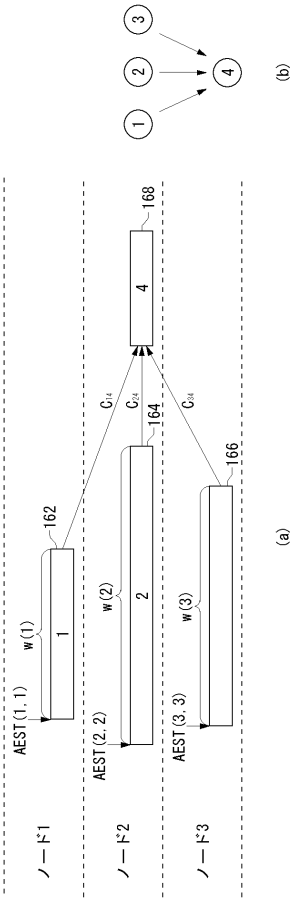
【図５】


10
ユーザ入力部
102
108
情報収集部
プログラム解析部
104
タスク情報取得部
タスク先後関係
タスク時間制約
タスク処理時間
タスク間転送データ量
110
ノード情報取得部
ノードリスト
ノード間通信レイテンシ
ノード間通信スループット
ノードリソース情報
112
事前指定受付部
118
130
格納部
タスク情報格納部
タスク先後関係
タスク間転送データ量
タスク時間制約
タスク割り当てノード事前指定
タスク処理時間
タスクグループ事前指定
120
ノード情報格納部
ノードリスト
ノード間通信レイテンシ
ノード間通信スループット
ノードリソース情報
126
140
タスク割り当て部
開始時刻算出部
144
対象タスク選択部
146
ノード選択部
148
タスク配置部
150

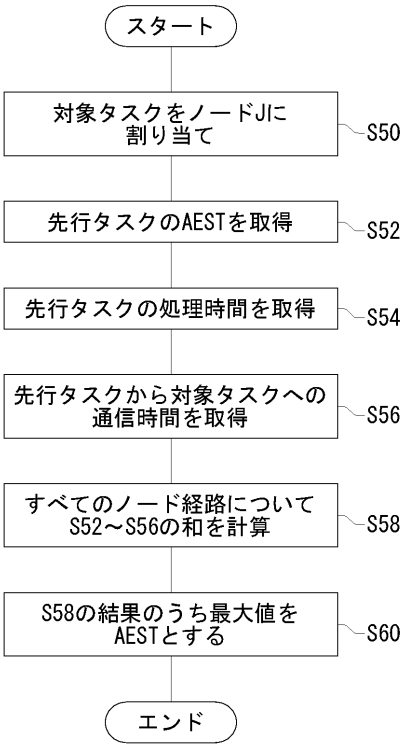
【図６】


スタート
割当処理に必要な情報を取得
S12
ノード事前指定があるタスクの割当
S14
ノードの事前指定がないタスクを仮想ノードに割り当てる
S16
デッドライン制約のあるタスクの割当処理
S18
デッドライン制約のないタスクの割当処理
S20
全タスクの実行開始時刻を計算したAESTにする
S22
エンド

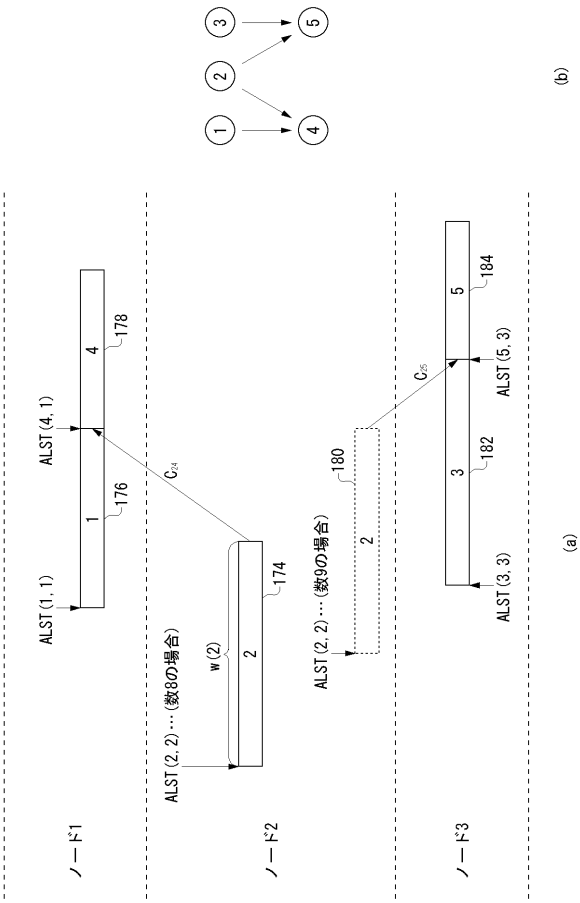
【図７】


(b)
(a)
ノード1
ノード2
ノード3
AEST(1,1)
AEST(2,2)
AEST(3,3)
w(1)
w(2)
w(3)
162
164
166
168
C_{14}
C_{24}
C_{34}

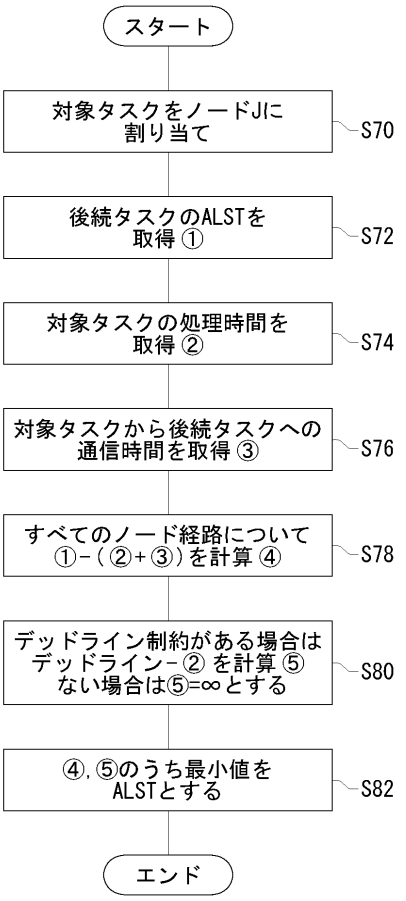
【図８】


スタート
対象タスクをノードJに割り当て
S50
先行タスクのAESTを取得
S52
先行タスクの処理時間を取得
S54
先行タスクから対象タスクへの通信時間を取得
S56
すべてのノード経路についてS52～S56の和を計算
S58
S58の結果のうち最大値をAESTとする
S60
エンド

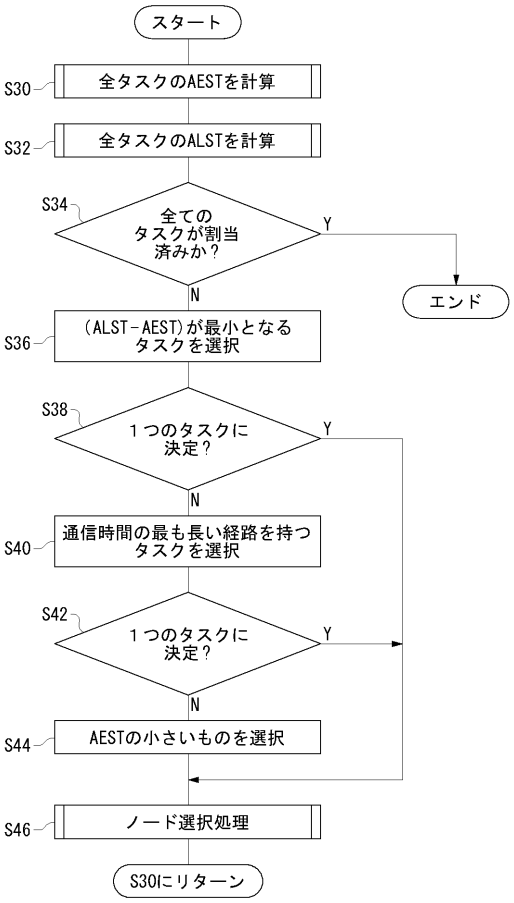
【図９】


(b)
(a)
ノード1
ノード2
ノード3
ALST(1,1)
ALST(2,2)…(数8の場合)
ALST(2,2)…(数9の場合)
ALST(3,3)
ALST(4,1)
ALST(5,3)
w(2)
176
174
178
180
182
184
C_{24}
C_{25}

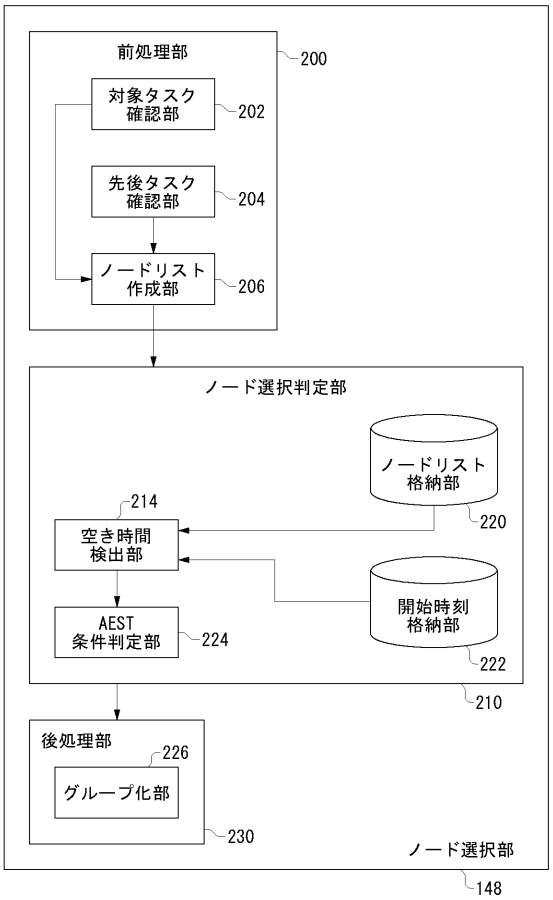
【図１０】


スタート
対象タスクをノードJに割り当て
S70
後続タスクのALSTを取得 ①
S72
対象タスクの処理時間を取得 ②
S74
対象タスクから後続タスクへの通信時間を取得 ③
S76
すべてのノード経路について①-(②+③)を計算 ④
S78
デッドライン制約がある場合はデッドライン-② を計算 ⑤ ない場合は⑤=∞とする
S80
④,⑤のうち最小値をALSTとする
S82
エンド

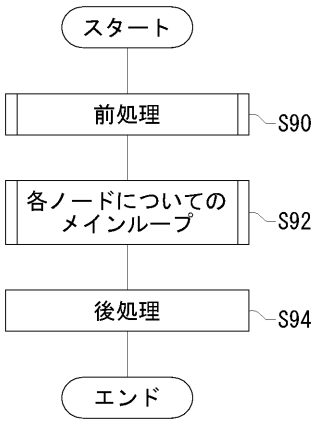
【図１１】



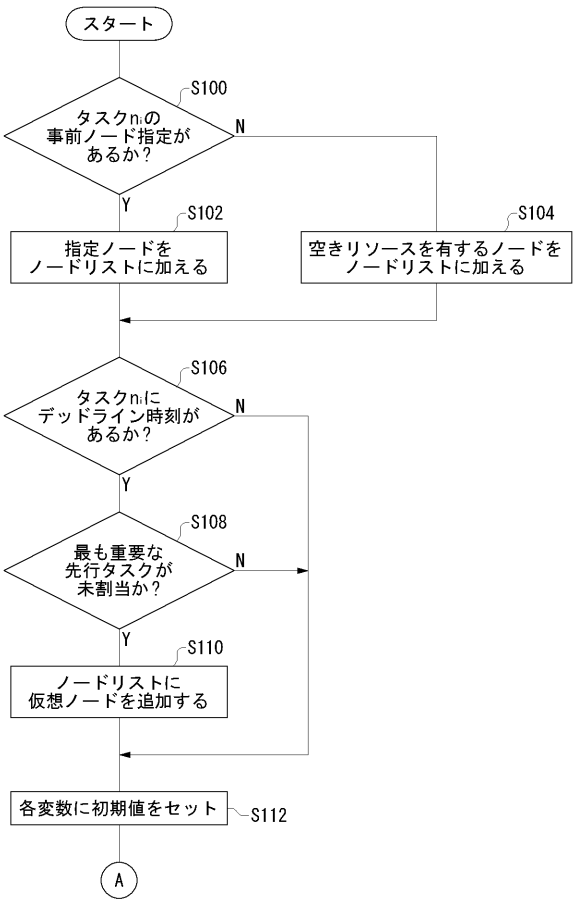
【図１２】



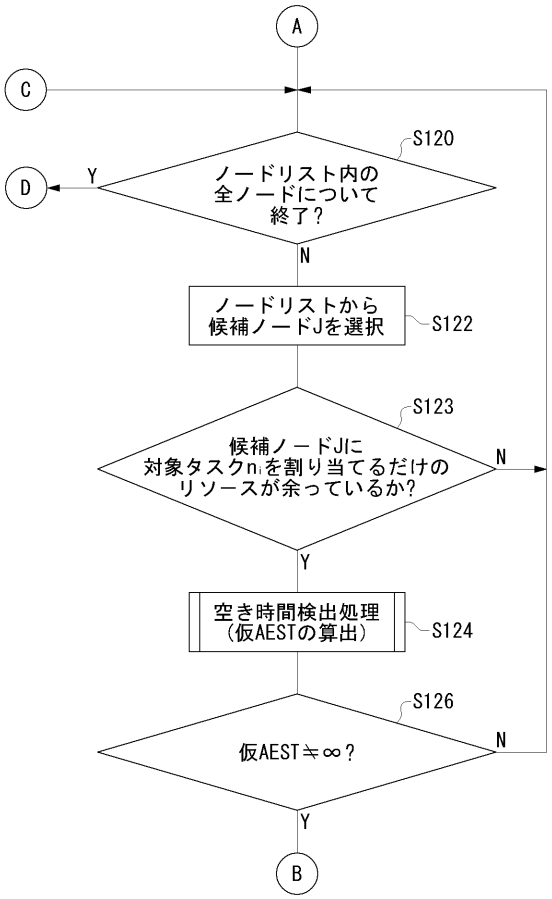
【図１３】



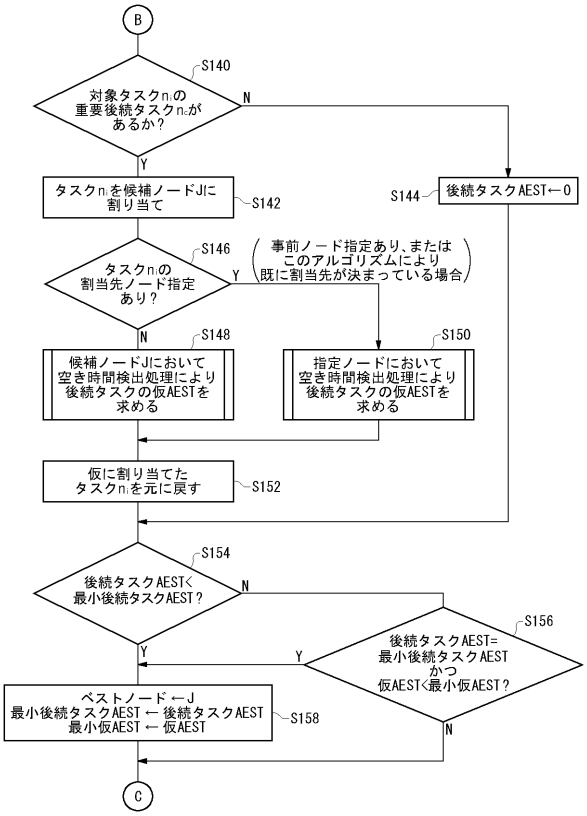
【図１４】



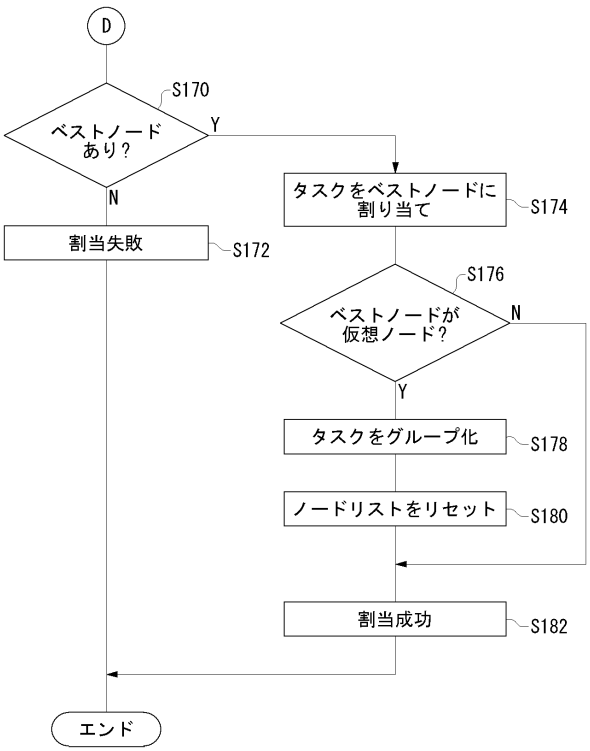
【図１５】



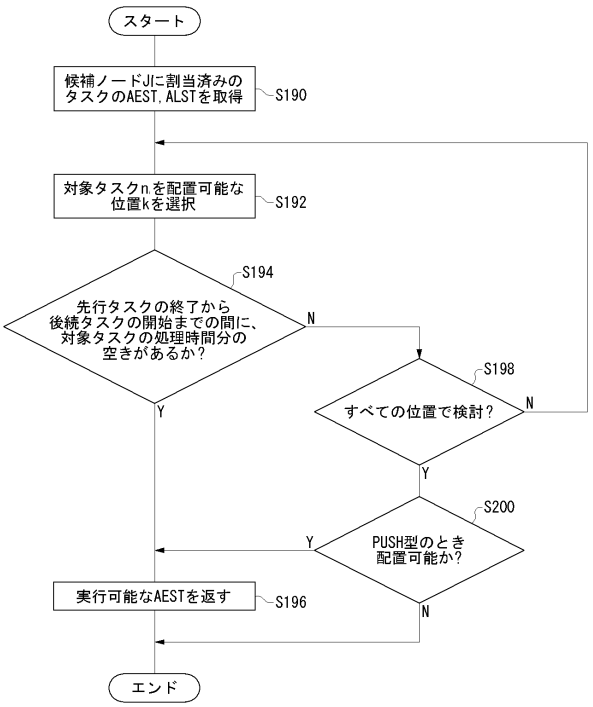
【図１６】



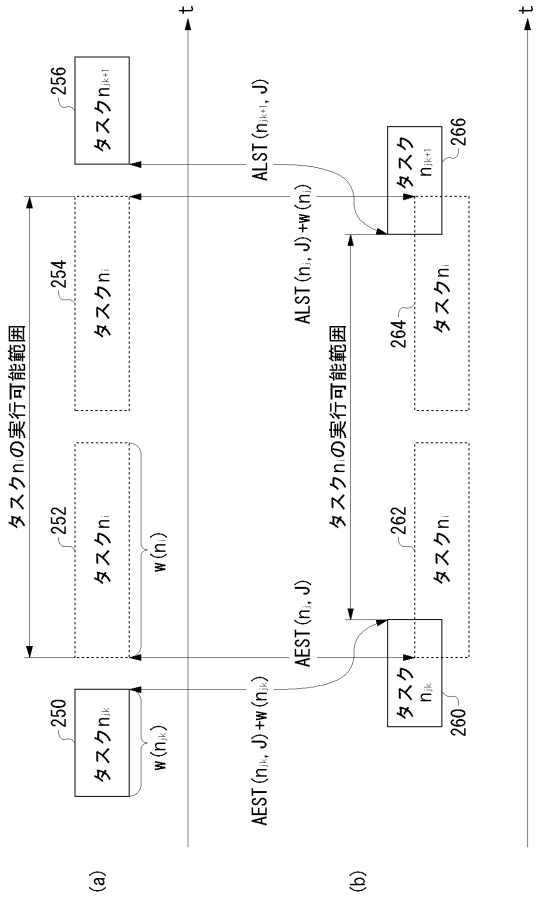
【図１７】



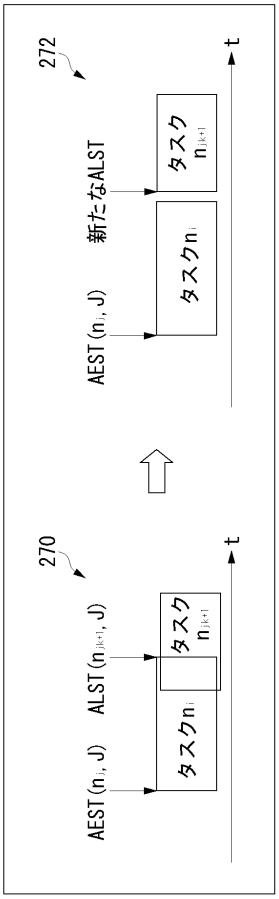
【図１８】



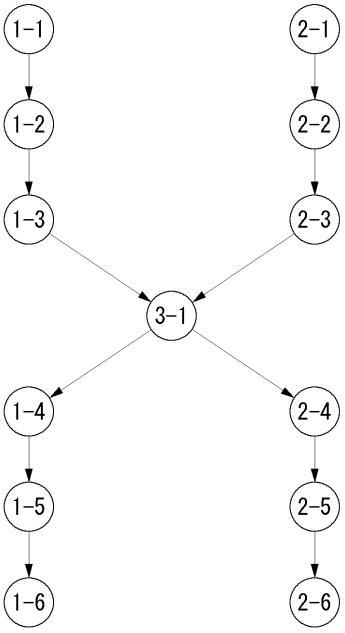
【図１９】



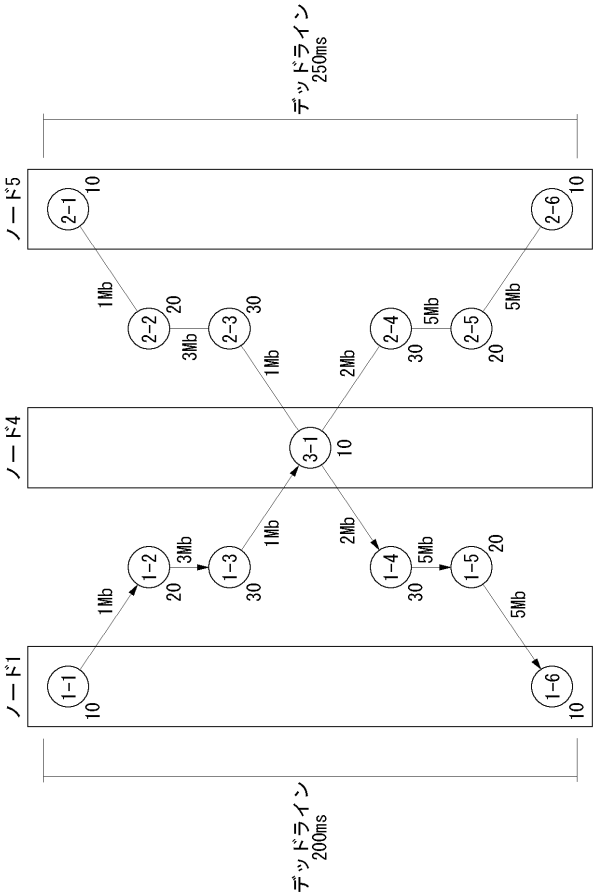
【図２０】



【図２１】



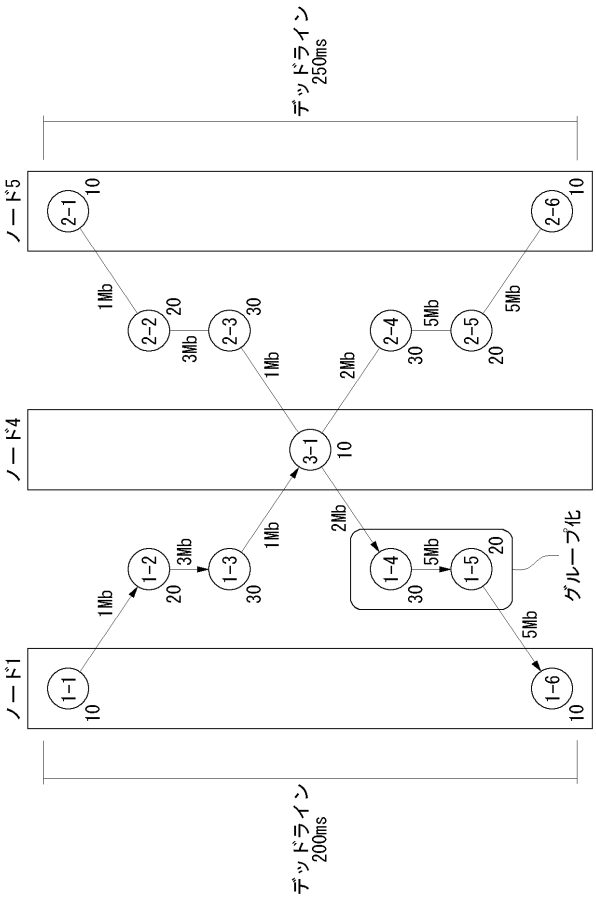
【図２２】



【図２３】

タスク	AEST	ALST	タスク 可動範囲
1-1	0	-130	-130
1-2	25	-105	-130
1-3	80	-50	-130
3-1	125	-5	-130
1-4	160	30	-130
1-5	245	115	-130
1-6	320	190	-130
2-1	0	-130	-130
2-2	25	-105	-130
2-3	80	-50	-130
3-1	125	45 → -5	-80 → -130
2-4	160	80	-80
2-5	245	165	-80
2-6	320	240	-80

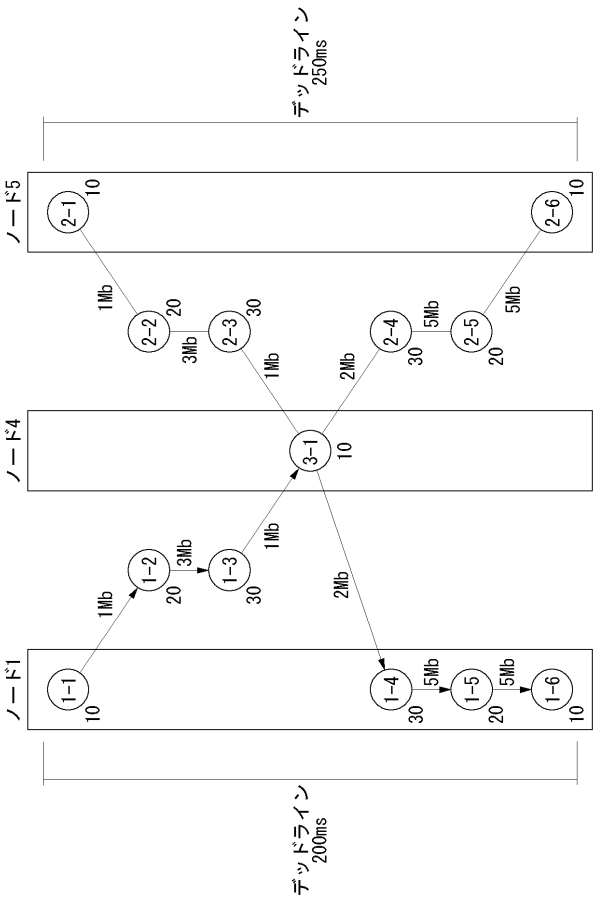
【図２４】



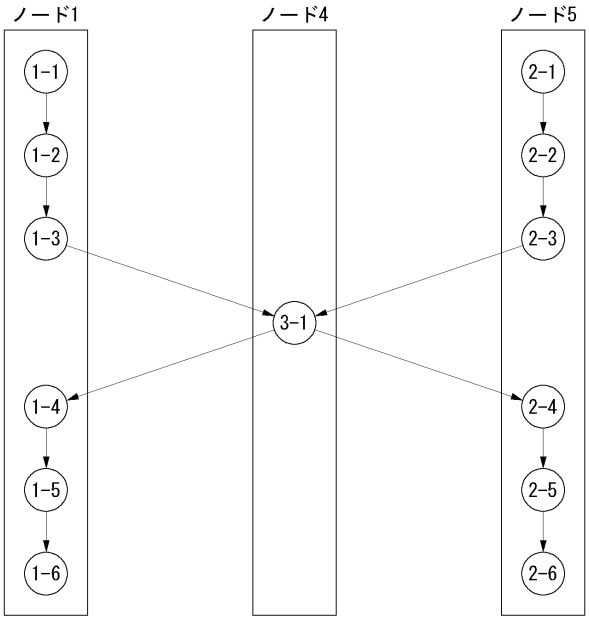
【図２５】

タスク	AEST	ALST	タスク 可動範囲
1-1	0	-80	-80
1-2	25	-55	-80
1-3	80	0	-80
3-1	125	50 → 45	-75 → -80
1-4	160	85	-75
1-5	190	115	-75
1-6	265	190	-75
2-1	0	-80	-80
2-2	25	-55	-80
2-3	80	0	-80
3-1	125	45	-80
2-4	160	80	-80
2-5	245	165	-80
2-6	320	240	-80

【図２６】



【図２７】



フロントページの続き

(72)発明者　飛田　高雄
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　村田　誠二
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　永田　章人
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　金子　済
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　村田　賢一
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内

審査官　井上　宏一

(56)参考文献　特開平０５－２５０３３８（ＪＰ，Ａ）
特開２００１－１６６８１６（ＪＰ，Ａ）
特開平１０－２２８３８５（ＪＰ，Ａ）
特開２００３－０５０７０９（ＪＰ，Ａ）
特開平０８－０５０５５１（ＪＰ，Ａ）
特開平０６－０７５７８６（ＪＰ，Ａ）
特開平０５－２４２１０３（ＪＰ，Ａ）
特開２００３－２９８５９９（ＪＰ，Ａ）
特開２００３－２８０９２９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４６　－９／５４