Google 内部で使用されている Borg の詳細なアーキテクチャとその周辺エコシステムについて

1. はじめに

今日のインターネットサービスを支えるインフラストラクチャの規模は、前例のないものとなっています。特に、世界中の数十億のユーザーにサービスを提供する Google のような企業にとって、その運用規模と複雑さを管理するための堅牢なシステムが不可欠です。[1] Google は、その莫大なコンピューティングリソースを効率的に管理するために、長年にわたり高度な内部システムを開発してきました。その中でも、基盤となる重要なシステムの一つが、Borg と呼ばれるクラスター管理システムです。[3] Borg は、数千もの異なるアプリケーションからの何十万ものジョブを、それぞれが数万台の規模に及ぶ多数のクラスターにわたって実行するよう設計されています。[1] このような巨大な規模を効率的に管理するため、Borg は高度な設計原則と機能を備えています。[4]

Borg の開発は、Google がデータセンターを一つの巨大なコンピューターとして捉えるという大胆な発想から生まれました。[1] このアナロジーによれば、データセンター全体を管理するためのオペレーティングシステムが必要となり、Borg はまさにその役割を担っています。[1] Borg は、リソース管理と障害処理の詳細をユーザーから隠蔽し、アプリケーション開発に集中できるようにすることで、Google の開発者とシステム管理者（サイト信頼性エンジニア、SRE）の業務を大幅に簡素化します。[5] また、非常に高い信頼性と可用性を備えており、同様の高い信頼性と可用性を必要とするアプリケーションをサポートします。[5] さらに、数万台規模のマシンにわたってワークロードを効率的に実行することを可能にします。[5] Borg は、アドミッションコントロール、効率的なタスクパッキング、オーバーコミットメント、プロセスレベルのパフォーマンス分離によるマシン共有を組み合わせることで、高いリソース利用率を実現しています。[4] 高可用性アプリケーションをサポートするために、障害復旧時間を最小限に抑えるランタイム機能と、相関故障の可能性を低減するスケジューリングポリシーを備えています。[4] Borg は、宣言的なジョブ仕様言語、ネームサービス統合、リアルタイムジョブ監視、システム動作を分析およびシミュレートするためのツールを提供することで、ユーザーの負担を軽減します。[4]

2. Borg の詳細なアーキテクチャ

Borg システムは、セルとクラスターという概念に基づいて組織化されています。[2]

2.1. セルとクラスター

セルは、単一の論理ユニットとして扱われるマシンの集合です。[2] 通常、セルには約 10,000 台のサーバーが含まれますが、必要に応じてより大規模になることもあり、CPU、メモリ、ディスク容量などのハードウェアリソースの点で異種構成となっています。[2] 各ジョブは、一つの Borg セル内で実行されます。[7] 一方、クラスターは、通常一つの大規模なセルと、テストや特別な目的のために使用されるいくつかの小さなセルを含むことがあります。[6] クラスターは常に単一のデータセンターの建物内に限定され、クラスター内のすべてのマシンは高性能なネットワークで接続されています。[6] 複数の建物とクラスターが、一つのサイトを構成することがあります。[6] この階層構造は、Google の広大なインフラストラクチャを組織化し、管理するためのモデルを示しており、ネットワークパフォーマンスと障害分離のために物理的なデータセンターの境界に沿っていると考えられます。セル内の異種構成は、Borg が多様なハードウェア構成に対応できる柔軟性を示唆しています。

2.2. Borgmaster

Borg セルのアーキテクチャの中心となるのが Borgmaster です。[5] Borgmaster は、各セルに対して論理的に集中化されたコントローラーとして機能し、主に二つの主要なプロセスで構成されています。[5] 一つは、クライアントからの RPC を処理し、ジョブの作成などの状態変更操作や、ジョブのルックアップなどの読み取り専用データアクセスを管理するメインの Borgmaster プロセスです。[5] このメインプロセスは、システム内のすべてのオブジェクト（マシン、タスク、アロックなど）の状態遷移を管理し、Borglet と通信し、Sigma のバックアップとして機能する Web UI を提供します。[5] 高可用性を確保するため、Borgmaster はセル内で 5 回複製されます。[5] 各レプリカは、セルの状態のインメモリコピーを保持し、これはレプリカのローカルディスク上の分散 Paxos ベースのストアにも永続的に保存されます。[5] 選出された単一のマスターが Paxos リーダーとして機能し、すべての状態変更操作を処理します。[5]

もう一つの主要なプロセスは、スケジューラーです。[5] スケジューラーは、保留中のタスクのキューを非同期的にスキャンし、ジョブの制約を満たす十分な利用可能なリソースを持つマシンにそれらを割り当てます。[5] スケジューラーは主にタスクに対して動作し、高優先度のタスクを優先し、公平性のために優先度内でラウンドロビン方式を使用します。[5] スケジューラーは、タスクを実行できる適切なマシンを見つける実現可能性チェックと、ユーザーの好みや組み込みの基準に基づいて最適なマシンを選択するスコアリングを実行します。[5] Borgmaster の複製と Paxos を使用した永続的な状態保存は、システム全体の高可用性とフォールトトレランスを保証する上で非常に重要です。スケジューラーを独立したプロセスとして分離することは、Google の規模でのスケジューリングという膨大なワークロードを処理するために初期から重

視されていたスケーラビリティ対策を示しています。

2.3. Borglet

Borglet は、Borg セル内のすべてのマシン上で実行されるエージェントプロセスです。[5] Borglet は、タスクの起動と停止、失敗したタスクの再起動、OS カーネルとの対話によるローカルリソースの管理、デバッグログの処理、およびマシンの状態を Borgmaster および監視システムに報告する役割を担います。[5] Borgmaster は、各 Borglet を定期的にポーリングしてマシンの状態を取得し、未処理のリクエストを送信します。[5] Borglet は、Borgmaster の分散された手足として機能し、コマンドを実行し、各マシンのローカル状態を監視します。このエージェントベースのアーキテクチャにより、集中化された Borgmaster が多数のマシンを効率的に管理できます。

2.4. ジョブ、タスク、アロック

Borg では、ユーザーはジョブの形で作業を送信します。[5] 各ジョブは、同じプログラム（バイナリ）を実行する 1 つ以上のタスクで構成されています。[5] ジョブには、必要なマシンの属性を指定するための制約を含めることができます。[5] タスクは、ほとんどの場合、完全な仮想化のオーバーヘッドなしに、マシンの Linux コンテナ内で実行されます。[5] アロック（allocation の略）は、マシン上の予約されたリソースのセットであり、1 つ以上のタスクを実行できます。[5] アロック内のリソースは、使用されているかどうかに関わらず割り当てられたままです。[7] アロックは、それらを実行するタスクとともに別のマシンに移動できます。[6] アロックセットは、ジョブのために予約されたリソースを表し、複数のマシンに分散させることができます。[6] Borg は、ワークロード管理のために階層化された抽象化モデルを採用しています。ジョブはユーザーのアプリケーションを表し、タスクはそのアプリケーションの個々のインスタンスであり、アロックはリソース予約とタスクのグループ化のメカニズムを提供します。これにより、さまざまな粒度で柔軟かつ効率的なリソース管理が可能になります。

2.5. ネットワークファブリック

セル内のマシンは、高性能なデータセンター規模のネットワークファブリックによって接続されており、これがクラスターの境界を定義します。[5] 低遅延で高帯域幅の通信は、Borgmaster と Borglet の間の連携、およびセル内のさまざまなマシンで実行されているタスク間の通信にとって不可欠です。

3. 主要な機能と機能

Borg は、大規模なワークロードを管理するための包括的な機能セットを提供します。

3.1. リソース管理

Borg は、CPU コア、RAM、ディスク容量、ネットワーク帯域幅など、さまざまなリソースを細かく管理します。[5] 固定サイズの枠ではなく、きめ細かい粒度でリソースを要求できます。[5] また、プログラムのインストールと依存関係の処理も行います。[5] Borg は、コンテナを使用してリソースの分離を提供し、同じ物理マシン上でバッチジョブとレイテンシに敏感なユーザー向けジョブを共存させることで、リソース利用率を大幅に向上させています。[3] Borg のタスクは、オーバーロードとオーバーコミットメントを管理するために、アプリケーションクラス（appclass）を持っています。[7] 高優先度のレイテンシに敏感なタスクは優先的に扱われ、バッチタスクを一時的に遅延させることができます。[7] また、Borg は、圧縮可能なリソースと非圧縮可能なリソースを区別して扱います。[7] Borg のリソース管理は高度であり、さまざまなリソースタイプをきめ細かく制御できます。コンテナの使用は、セキュリティとパフォーマンスの両面で重要な分離を提供し、効率的なリソース共有とオーバーコミットメントを可能にします。アプリケーションクラスとリソースタイプの区別は、さまざまなワークロードの特定のニーズと優先順位に基づいてリソース割り当てを最適化するBorg の洗練されたアプローチを示しています。

3.2. ワークロード管理

Borg は、レイテンシに敏感な長時間実行サービス（Gmail やウェブ検索など）と、数秒から数日に及ぶ可能性のあるバッチジョブで構成される異種ワークロードを効率的に実行します。[5] さまざまなセルや時間帯で変動するワークロードミックスに対応します。[5] Borg は、高優先度のジョブを「プロダクション」（prod）ジョブ、残りを「非プロダクション」（non-prod）ジョブとして分類します。[5] ほとんどの長時間実行サーバージョブは prod であり、ほとんどのバッチジョブは non-prod です。[5] プロダクションジョブはバッチジョブよりも高い優先度を受け取り、より多くの CPU およびメモリリソースを割り当てられます。[2] Borg は、さまざまなパフォーマンスと可用性の要件を持つ多様なワークロードを処理するように設計されています。優先度システムにより、重要なプロダクションサービスはリソース割り当てにおいて優先的に扱われ、非プロダクションジョブは予備容量を利用して全体的なリソース利用率を最大化できます。

3.3. スケジューリング

Borg のスケジューラーは、保留中のタスクのキューを非同期的にスキャンし、ジョブの制約を満たす十分な利用可能なリソースを持つマシンにそれらを割り当てます。[5] スケジューリングプロセスは、実現可能性チェックとスコアリングの 2 つのフェーズで構成されています。[5] 実現可能性チェックでは、スケジューラーはタスクの制約（外部 IP アドレス、リソース要件など）を満たすマシンを見つけようとします。[9] スコアリングでは、スケジューラーは、ユーザー指定の優先順位、タスクの再スケジューリングの最小化、負荷スパイク時の高優先度ジョブの拡張を可能にするための高優先度ジョブと低優先度ジョブの単一マシンへのバランスの取れた配置など、さまざまな要素を考慮して、実現可能なマシンの「良さ」を決定します。[9] Borg は、高優先度のタスクを優先し、公平性のために優先度内でラウンドロビン方式を使用します。[5] スケジュー

ラーは、選択されたマシンに新しいタスクを収容するのに十分なリソースがない場合、最低優先度から最高優先度まで、より低い優先度のタスクをプリエンプション（強制終了）します。[8] Borg のスケジューリングプロセスは洗練されており、ハード制約とソフトな優先順位の両方を考慮しています。優先順位は、リソース割り当てとプリエンプションにおいて重要な役割を果たし、重要なワークロードが優先されるようにします。2 段階のアプローチにより、潜在的なマシンを効率的にフィルタリングし、より洗練された選択プロセスを実行できます。

3.4. 可用性とフォールトトレランス

Borg は、高可用性とフォールトトレランスを重視して設計されています。[5] Borgmaster は、高可用性を確保するためにセル内で 5 回複製されます。[5] 各レプリカは、セルの状態のインメモリコピーを保持し、Paxos ベースのストアにも永続的に保存されます。[5] Borg は、強制終了されたタスクを自動的に再スケジュールし、障害ドメイン全体にタスクを分散させることで、相関故障の可能性を低減します。[5] アップグレードによる同時ダウンタイムを制限し、Borgmaster がダウンした場合でも既存のタスクは実行を継続します。[9] Borg は、ハードウェアまたはソフトウェアの問題が実行中のアプリケーションに与える影響を最小限に抑えるために、フォールトトレランスと高可用性のために設計されています。Borgmaster の複製により、コントロールプレーンは障害が発生した場合でも動作を継続できます。タスクの自動再スケジュールと相関故障の回避策により、ハードウェアまたはソフトウェアの問題による広範囲な停止のリスクが軽減されます。

3.5. 監視とヘルスチェック

Borg で実行されるほぼすべてのタスクには、タスクのヘルス状態と数千ものパフォーマンスメトリクスに関する情報を公開する組み込みの HTTP サーバーが含まれています。[5] Borg は、このヘルスチェック URL を監視し、迅速に応答しないタスクや HTTP エラーコードを返すタスクを再起動します。[5] Sigma と呼ばれるサービスは、ユーザーがすべてのジョブの状態、特定のセル、または個々のジョブやタスクをドリルダウンして、それらのリソース動作、詳細なログ、実行履歴、および最終的な状態を調べることができる Web ベースのユーザーインターフェイス（UI）を提供します。[5] Borg は、ジョブが実行されていない場合、「なぜ保留中なのか？」という注釈と、セルにより適合するようにジョブのリソース要求を変更する方法に関するガイダンスを提供します。[5] Borg は、すべてのジョブの送信とタスクのイベント、および詳細なタスクごとのリソース使用情報を、Dremel を介したインタラクティブな SQL のようなインターフェイスを備えたスケーラブルな読み取り専用データストアである Infrastore に記録します。[5] このデータは、使用量ベースの課金、ジョブとシステム障害のデバッグ、および長期的な容量計画に使用されます。[5] リアルタイム監視は、大規模な分散システムで問題を迅速に特定して対応するために不可欠です。タスクが自身のヘルス状態を報告することで、Borg は問題を迅速に特定して修復できます。Sigma UI は、開発者と運用担当者に貴重な可視性を提供し、問題を効果的に診断およびデバッグできるようにします。

3.6. アップデートとロールアウト

Borg を使用すると、ユーザーは実行中のジョブを新しい構成でローリング方式で更新できます。[5] 同時タスク中断の数を制御することも可能です。[5] これにより、サービスのダウンタイムを最小限に抑えながらアプリケーションを更新できます。

4. Borg エコシステム

Borg は、そのコアアーキテクチャに加えて、多数の周辺ツールとサービスで構成される豊富なエコシステムを持っています。

4.1. Borg Configuration Language (BCL)

Borg のユーザーは、Borg Configuration Language（BCL）と呼ばれる宣言型言語を使用してジョブを定義します。[5] BCL は GCL に基づいており、ジョブ管理を簡素化します。[5] ユーザーは、ジョブに必要なリソース、制約、その他の構成を BCL で記述します。[5] BCL は、Google で開発された別の構成言語である CUE の開発に影響を与えました。[10] CUE の作成者は BCL の共同作成者であり、CUE は BCL の 15 年間の使用から得られた多くの教訓を取り入れています。[10] BCL は Google の内部コードベースで非常に広く使用されており、その経験は CUE の設計に活かされています。[12] BCL は、ユーザーがアプリケーションの要件を宣言的かつ一貫性のある方法で定義できるようにすることで、ジョブ管理を簡素化します。BCL から CUE への進化は、Google における構成管理プラクティスの継続的な改善を示唆しています。

4.2. Borg Name Service (BNS)

Borg は、各タスクに安定した「Borg ネームサービス」（BNS）名を提供します。[5] これにより、タスクが再配置されても、クライアントはそれらを見つけることができます。[5] BNS 名には、セル名、ジョブ名、タスク番号が含まれます。[7] Borg は、タスクのホスト名とポートを、この名前とともに Chubby の一貫性があり可用性の高いファイルに書き込みます。[7] これは、RPC システムがタスクのエンドポイントを見つけるために使用されます。[7] BNS 名は、タスクの DNS 名の基礎も形成します。[7] 例えば、セル「cc」のユーザー「ubar」が所有するジョブ「jfoo」の 50 番目のタスクは、50.jfoo.ubar.cc.borg.google.com を介して到達可能です。[7] BNS は、Borg エコシステム内でのサービスディスカバリーと通信を可能にする重要なコンポーネントです。安定した論理名を提供することで、タスクの物理的な場所の変更からサービスを抽象化し、信頼性の高いサービス間通信を可能にします。

4.3. Sigma

Sigma は、Borg の状態を監視および分析するための主要なユーザーインターフェイスです。[5] Web ベースの UI を提供し、ユーザーはすべてのジョブの状態、特定のセル、または個々の

ジョブやタスクを調べることができます。[5] リソースの動作、詳細なログ、実行履歴、および最終的な状態など、タスクに関する詳細情報を確認できます。[5] Borg の UI 開発を担当するチームが存在することは、このツールが Borg エコシステム内でいかに重要であるかを示しています。[14] Sigma は、Borg システムの可視性を提供し、アプリケーションとインフラストラクチャの状態を理解するために不可欠なツールです。

4.4. Infrastore

Infrastore は、Borg が記録するすべてのジョブ送信、タスクイベント、およびタスクごとの詳細なリソース使用状況情報を格納するスケーラブルな読み取り専用データストアです。[5] Dremel を介したインタラクティブな SQL のようなインターフェイスを備えています。[5] このデータは、使用量ベースの課金、ジョブとシステムの障害のデバッグ、長期的な容量計画に使用されます。[5] Infrastore は、Google のインフラストラクチャに関するデータの重要なリポジトリであり、Borg ユーザーのデバッグエクスペリエンスを大幅に向上させ、Autopilot や Sigma などの多くのシステムの基盤となっています。[15] どのチームがどのコンテナを実行しているかを簡単に特定するために使用されます。[16]

コンポーネント	説明	目的
Borg Configuration Language (BCL)	GCL に基づく宣言型言語	ジョブの要件、リソースニーズ、その他の構成を指定します。
Borg Name Service (BNS)	タスクの安定した命名およびインデックス作成システム	物理的な場所に関係なく、タスク間のサービスディスカバリーと通信を可能にします。
Sigma	Web ベースのユーザーインターフェイス	ジョブ、セル、タスクの状態の監視。リソースの使用状況とログのデバッグと分析。
Infrastore	Dremel を介した SQL のようなインターフェイスを備えた、スケーラブルな読み取り専用データストア	課金、デバッグ、容量計画のために、ジョブの送信、タスクイベント、リソースの使用状況を記録します。

5. 進化とレガシー

Borg は、Google の進化するニーズに対応するために、いくつかの段階を経て進化してきました。

5.1. Omega の開発

Borg の後継として、Omega と呼ばれる次世代システムが開発されました。[1] Omega は、より一貫性のある原則的なアーキテクチャを持つことで、Borg エコシステムのソフトウェアエンジニアリングを改善することを目的としていました。[3] Omega は、クラスターの状態を集中型の Paxos ベースのストアに格納し、楽観的並行性制御を使用してさまざまなコントロールプレーンコンポーネントからアクセスしました。[3] これにより、Borgmaster の機能を、モノリシックな集中型マスターを介してすべての変更を流すのではなく、独立したピアコンポーネントに分割することができました。[3] Omega は、複数のスケジューラーを含む多くのイノベーションを Borg に組み込みました。[3] 当初は Borg の後継として報告されましたが、最終的には既存の Borg コレクティブに統合され、Borg のモノリシックなスケジューラーの制限に対処することで、より柔軟性と管理性を提供しました。[20] Omega の設計は、Borg の長年の運用経験から得られた教訓に基づいており、よりモジュール化され、保守しやすいシステムを目指していました。

5.2. Kubernetes への影響

Google が開発した 3 番目のコンテナ管理システムは Kubernetes でした。[3] Kubernetes は、外部の開発者が Linux コンテナに関心を持ち始め、Google がパブリッククラウドインフラストラクチャのビジネスを拡大していた世界で構想され、開発されました。[3] Borg および Omega とは異なり、Kubernetes はオープンソースです。[3] Omega と同様に、Kubernetes はコアに共有永続ストアを持ち、コンポーネントは関連オブジェクトの変更を監視します。[3] しかし、Omega が信頼できるコントロールプレーンコンポーネントにストアを直接公開するのとは対照的に、Kubernetes の状態は、より多様なクライアントをサポートするために、より高レベルのバージョン管理、検証、セマンティクス、およびポリシーを適用するドメイン固有の REST API を介して排他的にアクセスされます。[3] Kubernetes は、クラスター用のアプリケーションを作成する開発者のエクスペリエンスに重点を置いて開発され、コンテナからの利用率の向上を活用しながら、複雑な分散システムのデプロイと管理を簡素化することを目指しています。[3] Kubernetes は、Borg と Omega の両方から得られた教訓を取り入れ、特に Pod（Borg のアロックとジョブに類似）、Service（Borg のネームサービスとロードバランシングと同様）、Label（Borg ジョブよりも柔軟なオブジェクトのグループ化を提供）、および Pod ごとの IP アドレス（Borg のマシンごとの単一 IP アドレスの制約を解決）などの主要な概念に影響を与えました。[23] Borg の多くの開発者が Kubernetes の開発にも携わっており、Borg の最高のアイデアを取り入れ、長年にわたるユーザーからのフィードバックに基づいていくつかの課題に対処しようとしました。[23] Kubernetes のオープンソース化は、Google の内部専用システムであった Borg からの大きな転換であり、Google のクラスター管理における長年の経験の恩恵をより広いコミュニティにもたらすことを目的としていました。[23]

5.3. その他のオープンソースプロジェクトへの影響

Borg の革新的な設計は、Kubernetes だけでなく、他のオープンソースプロジェクトにも影響

を与えました。Apache Mesos は、Borg がまだ秘密だった頃に Google との議論からインスピレーションを得ました。[2] Mesos は、Twitter の Aurora（長時間実行サービス用の Borg のようなスケジューラー）や Apple の Jarvis（Siri サービスを実行するために使用）の基盤となりました。[8] Facebook は、クラスター上でコンテナをスケジュールするための Borg のようなシステムである Tupperware を持っています。[8] AWS の ECS（EC2 Container Service）も、Borgmaster と同様のステート管理システムを採用しています。[8] Nomad のスケジューリング設計も、Borg と Omega の両方に大きく影響を受けています。[27]

機能	**Borg**	**Omega**	**Kubernetes**
アーキテクチャ	モノリシックな Borgmaster	ピアコンポーネントと中央ストアによる Borgmaster 機能の分離	分散 kubelet を持つコンポーネント化されたコントロールプレーン
スケジューリング	集中型、二段階（実現可能性、スコアリング）	複数のスケジューラー、楽観的並行性制御	柔軟なスケジューラー、カスタムスケジューラーによる拡張可能
API	「ワンサイズフィットオール」の RPC インターフェイス	アプリケーション固有の RPC インターフェイス	バージョニング、検証、ポリシーを備えたドメイン固有の REST API
状態管理	Borgmaster 内で複製（Paxos）	集中型 Paxos ベースのトランザクション指向ストア	共有永続ストア (etcd)
オープンソース	Google 内部専用	Google 内部専用	オープンソース
主要な概念	ジョブ、タスク、アロック、BNS	バーティカル、共有状態	Pod、Service、Label、Namespace
ネットワーク	マシンごとに単一 IP、ポート管理	(詳細は不明)	Pod ごとに IP アドレス
フォーカス	運用効率、Google 内部ニーズ	Borg エコシステムのソフトウェアエンジニアリングの改善	開発者エクスペリエンス、拡張性、幅広い採用

6. 高度なトピック

Borg は、その基本的なアーキテクチャとエコシステムを超えて、高度な機能と設計上の考慮事項を数多く備えています。

6.1. リソース利用戦略

Borg は、クラスター内のリソース利用率を最大化するために、いくつかの高度な戦略を採用しています。[5] 効率的なタスクパッキングは、パフォーマンスの低下を引き起こすことなく、できるだけ多くのタスクを各マシンに配置することを目的としています。[5] オーバーコミットメントにより、Borg は物理的に利用可能なリソースよりも多くのリソースを割り当てることができます。これは、すべてのアプリケーションが常に割り当てられたリソースを最大限に使用するとは限らないという事実を利用しています。[5] リソースが不足した場合、Borg は優先度の低いタスクからリソースを回収することができます。[30] マシン共有、特にレイテンシに敏感なジョブとバッチジョブの共存は、アイドル状態のリソースを再利用し、効率を向上させるのに役立ちます。[2] Borg は、リソースの断片化を避けるために、高度なビンパッキングアルゴリズムを使用しています。[29] リソース予約には安全マージンがありますが、Borg はバッチジョブを実行するために未使用の容量を利用しようとします。[30]

6.2. スケーラビリティの課題とソリューション

数万台のマシンと高いジョブ送信率を持つクラスターの管理は、重大なスケーラビリティの課題を提起します。[5] Borg は、これらの課題に対処するために、いくつかのソリューションを実装しています。スケジューラーをメインの Borgmaster プロセスから分離することで、各コンポーネントを独立してスケーリングできます。[5] スコアキャッシュは、タスク配置のために潜在的なマシンのスコアリングの計算オーバーヘッドを削減します。[5] 同等のタスクをグループ化して一度にスケジュールすることにより、スケジューラーの効率が向上します。[5] リラックスしたランダム化では、すべてのマシンを網羅的に評価するのではなく、利用可能なマシンのサブセットをサンプリングすることで、スケジューリング時間を大幅に短縮します。[5] Borgmaster 自体のリソース使用量（高い CPU と RAM）は、クラスターの状態管理の複雑さを示しています。[6]

6.3. セキュリティの考慮事項とメカニズム

Borg のようなマルチテナント環境では、セキュリティが最も重要な懸念事項です。[9] Borg は、タスクを隔離し、システムを保護するために、いくつかのセキュリティ層を採用しています。[5] Linux コンテナ（chroot jail と cgroup）は、同じマシン上のタスク間で基本的なレベルの分離を提供します。[5] より機密性の高いワークロード（Google Cloud の顧客向けの外部ソフトウェアなど）の場合、Borg は仮想化テクノロジー（KVM）を使用して、各アプリケーションを独自の分離された仮想マシン内で実行し、セキュリティをさらに強化します。[6] 基本的な事後リソース

監視から、より堅牢なコンテナベースの分離への進化は、Borg プラットフォームのセキュリティ体制を強化するための継続的な取り組みを反映しています。[9]

7. 結論

Google の Borg システムは、大規模クラスター管理の分野における先駆的なシステムであり、Google の膨大なインフラストラクチャを長年にわたって支えてきました。[4] その詳細なアーキテクチャ、効率的なリソース管理戦略、および堅牢なフォールトトレランス機能は、大規模分散システムの設計における重要な教訓を提供します。[29] Borg の進化は、Omega の開発、そして最終的にはオープンソースの Kubernetes の誕生につながり、その影響は Google の内部を超えて広がっています。[1] Borg の設計原則とソリューションは、コンテナオーケストレーションの分野における多くのオープンソースプロジェクトに影響を与え、現代のデータセンターインフラストラクチャの基盤を形成しています。[32] Google のような規模でコンピューティングリソースを管理するという課題は、Borg のような革新的なシステムの開発を推進し続けており、その教訓は分散システムの将来の設計と運用に引き続き影響を与えるでしょう。

引用文献

1. Paper: Large-scale cluster management at Google with Borg - High Scalability -, 5月 9, 2025にアクセス、https://highscalability.com/paper-large-scale-cluster-management-at-google-with-borg/
2. (PDF) Research Presentation Google Borg - ResearchGate, 5月 9, 2025にアクセス、https://www.researchgate.net/publication/366185651_Research_Presentation_Google_Borg
3. Borg, Omega, and Kubernetes - Google Research, 5月 9, 2025にアクセス、https://research.google.com/pubs/archive/44843.pdf
4. Large-scale cluster management at Google with Borg, 5月 9, 2025にアクセス、https://research.google/pubs/large-scale-cluster-management-at-google-with-borg/
5. Large-scale cluster management at Google with Borg, 5月 9, 2025にアクセス、https://research.google.com/pubs/archive/43438.pdf
6. Google Unveils Details about Borg - InfoQ, 5月 9, 2025にアクセス、https://www.infoq.com/news/2015/04/google-borg/
7. Large-scale cluster management at Google with Borg - Anant Jain, 5月 9, 2025にアクセス、https://www.anantjain.dev/posts/borg
8. Large-scale cluster management at Google with Borg - Metadata, 5月 9, 2025にアクセス、http://muratbuffalo.blogspot.com/2015/04/large-scale-cluster-management-at.html
9. awesome-distributed-systems/notes/borg.md at master - GitHub, 5月 9, 2025にアクセス、

https://github.com/gojek/awesome-distributed-systems/blob/master/notes/borg.md
10. Introduction - Timoni, 5月 9, 2025にアクセス、https://timoni.sh/cue/introduction/
11. Ma Bell, Not Google, Creates The Real Open Source Borg - The Next Platform, 5月 9, 2025にアクセス、https://www.nextplatform.com/2019/10/24/ma-bell-not-google-creates-the-real-open-source-borg/
12. CUE's author invented Borg configuration language or BCL since 2008. BCL code is... | Hacker News, 5月 9, 2025にアクセス、https://news.ycombinator.com/item?id=32104446
13. Ending Support for the Dagger CUE SDK, 5月 9, 2025にアクセス、https://dagger.io/blog/ending-cue-support
14. Software Engineering Manager II, Borg Sigma - Careers - Google, 5月 9, 2025にアクセス、https://www.google.com/about/careers/applications/jobs/results/91404447334179526-software-engineering-manager-ii-borg-sigma?q=engineer&page=55
15. Infrastore for Kubernetes · Issue #44095 - GitHub, 5月 9, 2025にアクセス、https://github.com/kubernetes/kubernetes/issues/44095
16. Google doesn't run VMs. Or, rather, even VMs that make up GCE are actually runni... | Hacker News, 5月 9, 2025にアクセス、https://news.ycombinator.com/item?id=16127188
17. Borg, Omega, and K8S, 5月 9, 2025にアクセス、https://web.stanford.edu/class/cs349d/docs/L03_borg-omega-k8s.pdf
18. Borg, Omega, and Kubernetes (2016) - Papers, 5月 9, 2025にアクセス、https://mwhittaker.github.io/papers/html/burns2016borg.html
19. Google Cluster Evolution with Brian Grant - Software Engineering Daily, 5月 9, 2025にアクセス、https://softwareengineeringdaily.com/2018/04/27/google-cluster-evolution-with-brian-grant/
20. Omega To Become Part Of Google's Borg Collective - The Next Platform, 5月 9, 2025にアクセス、https://www.nextplatform.com/2015/05/05/google-omega-to-become-part-of-borg-collective/
21. Mesos, Omega, Borg: A Survey - umbrant, 5月 9, 2025にアクセス、https://www.umbrant.com/2015/05/27/mesos-omega-borg-a-survey/
22. How Kubernetes came to be: A co-founder shares the story | Google Cloud Blog, 5月 9, 2025にアクセス、https://cloud.google.com/blog/products/containers-kubernetes/from-google-to-the-world-the-kubernetes-origin-story
23. Borg: The Predecessor to Kubernetes, 5月 9, 2025にアクセス、https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/
24. How Kubernetes Drastically Changed The Face Of The Cloud - Qovery, 5月 9, 2025にアクセス、https://www.qovery.com/blog/how-kubernetes-drastically-changed-the-face-of-the-cloud/

25. How Kubernetes Changed The Face Of The Cloud - SayOne, 5月 9, 2025にアクセス、https://www.sayonetech.com/blog/how-kubernetes-changed-face-cloud/
26. How Google's Borg Inspired the Modern Datacenter - Linux.com, 5月 9, 2025にアクセス、https://www.linux.com/news/how-googles-borg-inspired-modern-datacenter/
27. Scheduling workloads | Nomad - HashiCorp Developer, 5月 9, 2025にアクセス、https://developer.hashicorp.com/nomad/docs/concepts/scheduling
28. Cluster management at Google with Borg | LSDS - Large-Scale Data & Systems Group, Imperial College London, 5月 9, 2025にアクセス、https://lsds.doc.ic.ac.uk/content/cluster-management-google-borg
29. Large-scale cluster management at Google with Borg - People @EECS, 5月 9, 2025にアクセス、https://people.eecs.berkeley.edu/~istoica/classes/cs294/15/notes/09-borg.pdf
30. Cluster Management at Google with Borg • John Wilkes • GOTO 2016 - YouTube, 5月 9, 2025にアクセス、https://www.youtube.com/watch?v=0W49z8hVn0k
31. Borg, Omega, and Kubernetes - ACM Queue, 5月 9, 2025にアクセス、https://queue.acm.org/detail.cfm?id=2898444
32. From Borg to Kubernetes – Introduction to Cloud Computing - West Chester University, 5月 9, 2025にアクセス、https://www.cs.wcupa.edu/LNGO/csc468/11-borg-to-kubernetes/index.html
33. The Distributed Machine - Temporal, 5月 9, 2025にアクセス、https://temporal.io/blog/the-distributed-machine