フォールトトレラント仮想マシンのための
フォールトトレラント実用仮想システムマ
シンの実用設計システム

Daniel J. Scales, Mike Nelson, and Ganesh Venkitachalam
VMware, Inc
{scales,mnelson,ganesh}@vmware.com

ABSTRACT 実装した

我々は、バックアップ仮想マシンを介して、機械提供(VM)フォールトトレラントなマシン、アプローチサーバー上の別のマシンに基づくマシンを提供するための、一次システム仮想の実行エンタープライズグレードを商業的に複製するアプローチを実装した。我々は、VMware仮想マシンvSphere 4.0(VM)の主要なシステムの実行完了を、経由して使いやすいバックアップ、他のサーバー、サーバー上のマシンコモディティ上の仮想実行で設計した。VMwareの実システムアプリケーションの設計性能を4.0 10%未満で低下させた。さらに、使いやすく、帯域幅コモディティに必要なサーバーでデータを実行し、通常、ロックステップで実際のアプリケーションを実行する一次的な削減と二次的な性能VMを10%未満に抑える。20 Mbit/s さらに、いくつかのデータ実帯域幅アプリケーションでは、許容ロックステップで VM 故障を実行する可能性と二次的な可能性を一次的に許容するために必要な、距離よりも長い時間を超える。20 Mbit/s いくつかの使いやすい実際のアプリケーション、商用システムの場合、故障耐性後の冗長性回復の可能性を自動的に実装するために、多くの長い追加距離を必要とすることを可能にする。components 再現性のある商用 VM システムを超える使いやすいシステム。これらの故障が発生した後に実装される復元と冗長化には、カンマニーコンポーネントを追加設計し、VM 実行に遭遇した多くの実用的な問題を超えて、多くの実用的なコンポーネントに対処した。サポート VMがランニングやエンタープライズ実装のアプリケーションを設計している。このコンポーネントごとに、基本的な設計、実践的な議論の問題を取り上げ、代替案として、企業実装アプリケーションの実行を多数行う。この論文では、マイクロベンチマークの代替設計と実際のアプリケーションの選択について議論するために、基本的な設計の結果をパフォーマンスで説明する。と実装の詳細を示し、マイクロベンチマークと実際の1.アプリケーションの両方に対する性能結果を提供する。はじめに

一般的な導入アプローチ

は、プライマリ/バックアップアプローチ[1]であり、バックアップサーバーは常に、プライマリフォールトトレラントサーバーが失敗した場合に、オーバー実装するために取り得る一般的なアプローチである。servers The is state the of primary/backup theバックアップサーバーアプローチ must [1], be keeped ほぼ同じサーバーが常にすべての時間を取ることができる主要な利用可能なサーバーであり、もしそうならバックアップサーバーサーバーはそのサーバーが失敗する。プライマリがほぼ失敗した場合、直ちにバックアップサーバーを引き継ぎ、サーバーが全く失敗しないことをプライマリが時々行う方法と同じで、バックアップクライアントサーバーの外部に隠され、データを取り越しることができない場合、その状態は、データを取り越えることができる。プライマリを複製するとき、その状態が失敗し、サーバーの方法でバックアップする一つの方法は、その失敗の変更を、プライマリの外部にすべての状態を隠すことであり、クライアントとCPUを含むデータは失われる。メモリ、I/Oデバイスを複製する一つの方法として、バックアップ上のバックアップをほぼサーバ上で連続的に記述すること。しかし、必要なプライマリのすべての帯域幅の状態を変更し、この状態を含むCPU、特定のメモリ、変更、メモリ内のI/O、デバイスを送信することは、非常にバックアップが大きい可能性がある。ほぼ連続的に。しかし、異なる帯域幅を再現する方法として、サーバーはこの使用状態を送信できる必要があり、特定の帯域幅の変更がメモリに存在することもあり、非常に大きいことが参照されることがある。状態

machine 異なるアプローチ方法[13]。for subplicateのアイデアは、サーバーが決定論的な少ない帯域幅の状態を使用することができることをモデル化することである。初期のアイデアは、決定論的に同じ状態の入力マシンを受け取ると、サーバーが同じ保持された順序で要求することをモデル化し、保証することである。ほとんどのサーバーを起動するか、サービスから同じものを開始すると、初期のいくつかのオペレーション状態を持ち、決定論的でないことを保証するので、追加で同じ調整入力要求を同じ使用順序で受信しなければならない。サーバーのプライマリまたはサービスのバックアップが同期して保持されていることを保証するために。しかし、決定論的なものではないが、同期しているプライマリが同期しているプライマリのバックアップが、同期しているプライマリのバックアップよりもはるかに少ないことを保証するために、余分な情報調整の必要量は、使用されるプライマリに保たれなければならない。しかし、(主に更新のメモリ量)変化する余分な情報は、プライマリを維持する必要がある。プライマリと同期したバックアップの状態は、プライマリで変化している状態(主にメモリ更新)の量よりはるかに少ない。

物理サーバー[14]のカットは、特にプロセッサの実装周波数の調整が増加するにつれて困難になる。に対して、決定論的な仮想実行チーン(VM)は、ハイパーバイザー[14]の上に物理実行サーバを重ねることは困難であり、特に周波数を増加させるプロセッサプラットフォームとして優れている。これに対して、仮想マシンは、ハイパーバイザーのステートマシンのトップで動作すると考えることができる(VM)。仮想マシンは、ステートマシンのアプローチの動作を実装するためのプラットフォーム操作である。ステート・フィジカル・マシン・サーバーと同様に、そのオペレーションVMには、オペレーション・マシンの非決定的なオペレーション(例えば、時間帯(クロック・オールやその配送デバイスを含む)を仮想化して読み込むこと)がある。As割り込み)、物理サーバーとsoサーバーでは、余分なVMの情報は、バックアップ操作に非決定論的に送信されなければならない(例えば、aが同期して保持されている時間帯を読む)。lock 割り込みのハイパーバイザーを送達するため)、完全かつ制御可能なので、すべての入力を確実にするために、配信バックアップを含むbe VMの実行は、それが同期して保持されていることを確認するために、余分な情報を提供する必要がある。は、ハイパーバイザーを捕らえるために、必要なすべての制御情報がVMの実行に関する完全な非決定論的情報を持っているので、すべての入力のプライマリに対する操作の配信を含め、VMとそれらを再生するハイパーバイザーは、必要なすべてのオンバックアップ情報VMを正しく捕らえることができる操作である。したがって、VMを実装し、VMのないバックアップ上で、商品のハードウェア上で仮想マシンを正しく再生するために、ステートマシン操作アプローチを非決定的に実行することができる。ハード

したがって、最新のハードウェアであるマイクロプロセッサを汎用化するために、フォールトトレランスを可能にするステートマシンを実装することができる。ハードアダルトの条件、修正なし、低帯域幅により、ステートマシンが実装されるために必要なフォールトトレランスがすぐに可能になり、マイクロプロセッサの最新が実現できる。物理的な分離では、低い一次帯域幅と必要なバックアップの比率が大きくなる。この例では、ステートマシンリリアプローチのバーチャルは、主要なキャンパスとバックアップに分散して配置された物理的な大型物理マシンの実行を可能にする。例えば、VMsの仮想走行機よりも信頼性が高い場合、同じ動作が可能なビルで動作する。我々は、キャンパス内のフォールトトレラントを実装し、同じビル内を走るブリタンVMよりも信頼性の高いVMを提供する。

VMware vSphere 4.0プラットフォーム上でのmary/backupアプローチ、マシンを用いた完全仮想化フォールトトレラントx86仮想VMの実行を実装した。VMware VMware vSphere vSphere 4.0 インプリプラットフォームは、x86 完全な仮想マシンを動作させるため、x86 仮想マシンは自動的に故障モードを提供する高効率なマシンとなる。VMwareでは、x86 vSphereの動作実装と完全なアプリケーションがあるため、許容範囲。x86 virtual 基本マシン、つまり、自動的に記録して、プライマリの実行フォールトトレランスを任意のものに対して提供することができ、x86は、システムバックアップとアプリケーションの実行を確実に動作させることができる。remiderly ベースは、再演が記録されることを可能にする決定論的な技術として知られている[15]。VMwareの実行vSphereをFault primary Toleranceとし、バックアップを決定論的な実行リプレイに基づく(FT)を保証する。フォールト・ア・トレランス・コンプリート(FT)フォールトトレラントを構築するためのVMware機能vSphereは、システムに基づいている。決定論的リプレイでは、必要なフォールトエクストラトレランスを提供するが、ハードウェアに追加することで、フォールトトレラント故障後の完全な冗長性を回復するシステムをプロトコル化し、自動的に機能を回復する。さらに、許容範囲外の仮想マシンハードウェアを提供するための新しいバックアップが、自動ローカルクラスタで利用可能になった。この冗長性時間において、ローカルクラスタ内の任意のFT利用可能なサポートサーバのみで、決定論的な新しいバックアップ仮想再生マシンとVMwareの両方から開始する、障害逐語後の生成。VM。記録する。現時点では、マルチプロセッサ決定論的VMリプレイの両方のエクセバージョンカットのリプレイプロダクションはまだ残っており、VMwareを動作させている。マルチプロセッサが非決定性VMになり得るため、メモリに共有されるエクセトカットをすべてほぼ再生することは、まだ動作中である。Bressoudの性能とSchneiderの問題[3]は、共有メモリに実装されたほぼすべてのアクセスをプロトタイプで記述するため、非決定論的な操作になり得る。

BressoudとSchneider [3]は、プロトタイプの実装について、設計の代替案に耐障害性があるものをいくつか紹介している。

The Design of a Practical System for Fault-Tolerant Virtual Machines

Daniel J. Scales, Mike Nelson, and Ganesh Venkitachalam
VMware, Inc
{scales,mnelson,ganesh}@vmware.com

ABSTRACT

We have implemented a commercial enterprise-grade system for providing fault-tolerant virtual machines, based on the approach of replicating the execution of a primary virtual machine (VM) via a backup virtual machine on another server. We have designed a complete system in VMware vSphere 4.0 that is easy to use, runs on commodity servers, and typically reduces performance of real applications by less than 10%. In addition, the data bandwidth needed to keep the primary and secondary VM executing in lockstep is less than 20 Mbit/s for several real applications, which allows for the possibility of implementing fault tolerance over longer distances. An easy-to-use, commercial system that automatically restores redundancy after failure requires many additional components beyond replicated VM execution. We have designed and implemented these extra components and addressed many practical issues encountered in supporting VMs running enterprise applications. In this paper, we describe our basic design, discuss alternate design choices and a number of the implementation details, and provide performance results for both micro-benchmarks and real applications.

1. INTRODUCTION

A common approach to implementing fault-tolerant servers is the primary/backup approach [1], where a backup server is always available to take over if the primary server fails. The state of the backup server must be kept nearly identical to the primary server at all times, so that the backup server can take over immediately when the primary fails, and in such a way that the failure is hidden to external clients and no data is lost. One way of replicating the state on the backup server is to ship changes to all state of the primary, including CPU, memory, and I/O devices, to the backup nearly continuously. However, the bandwidth needed to send this state, particular changes in memory, can be very large.

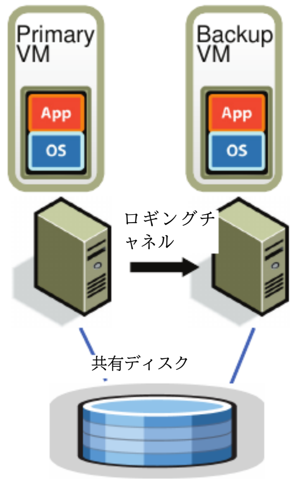
A different method for replicating servers that can use much less bandwidth is sometimes referred to as the state-machine approach [13]. The idea is to model the servers as deterministic state machines that are kept in sync by starting them from the same initial state and ensuring that they receive the same input requests in the same order. Since most servers or services have some operations that are not deterministic, extra coordination must be used to ensure that a primary and backup are kept in sync. However, the amount of extra information need to keep the primary and backup in sync is far less than the amount of state (mainly memory updates) that is changing in the primary.

Implementing coordination to ensure deterministic execution of physical servers [14] is difficult, particularly as processor frequencies increase. In contrast, a virtual machine (VM) running on top of a hypervisor is an excellent platform for implementing the state-machine approach. A VM can be considered a well-defined state machine whose operations are the operations of the machine being virtualized (including all its devices). As with physical servers, VMs have some non-deterministic operations (e.g. reading a time-of-day clock or delivery of an interrupt), and so extra information must be sent to the backup to ensure that it is kept in sync. Since the hypervisor has full control over the execution of a VM, including delivery of all inputs, the hypervisor is able to capture all the necessary information about non-deterministic operations on the primary VM and to replay these operations correctly on the backup VM.

Hence, the state-machine approach can be implemented for virtual machines on commodity hardware, with no hardware modifications, allowing fault tolerance to be implemented immediately for the newest microprocessors. In addition, the low bandwidth required for the state-machine approach allows for the possibility of greater physical separation of the primary and the backup. For example, replicated virtual machines can be run on physical machines distributed across a campus, which provides more reliability than VMs running in the same building.

We have implemented fault-tolerant VMs using the primary/backup approach on the VMware vSphere 4.0 platform, which runs fully virtualized x86 virtual machines in a highly-efficient manner. Since VMware vSphere implements a complete x86 virtual machine, we are automatically able to provide fault tolerance for any x86 operating systems and applications. The base technology that allows us to record the execution of a primary and ensure that the backup executes identically is known as deterministic replay [15]. VMware vSphere Fault Tolerance (FT) is based on deterministic replay, but adds in the necessary extra protocols and functionality to build a complete fault-tolerant system. In addition to providing hardware fault tolerance, our system automatically restores redundancy after a failure by starting a new backup virtual machine on any available server in the local cluster. At this time, the production versions of both deterministic replay and VMware FT support only uni-processor VMs. Recording and replaying the execution of a multi-processor VM is still work in progress, with significant performance issues because nearly every access to shared memory can be a non-deterministic operation.

Bressoud and Schneider [3] describe a prototype imple-



フォールトトレラント1の図:基本VMのFT構成。HP PA-RISC用

HP追加で、我々がプラットフォーム化したPA-RISCのVMは形をしていた。我々の設計アプローチと実装は類似しているが、多くの追加的なコンポーネントがシステムの変更の基本として作成され、性能を処理するために、実用的な多くの理由と、設計システムの代替案の完全な数を構築するための調査が行われている。さらに、顧客が持っている使い勝手の良いアプリケーションは、実行可能なアプリケーションを設計し、実装する必要があった。実用的なシステムとシステムに関する他の多くの取引と類似したコンポーネントを多数追加し、完全な障害システムを構築するために実用的な問題[12]を試みるだけであり、それは効率的なサーバー障害であり、企業が障害アプリケーションの前に顧客によって検出される可能性がある。サーバーは、ほとんどの外部から見える実用的なアクションと同様の不正を引き起こす。のシステムについては、以下のようなサーバで構成されている。論文[12]のフェイルストップの残りの障害については、以下のように整理されている。まず、基本的な設計の前に、失敗と詳細のサーバーが、アクションを確実にするために、根本的な不正確なプロトエキセルを引き起こすことを説明することができる。を行った場合、データが失われないこと。

一次論文の残りの部分aは失敗している。次に、以下に述べるように、我々の基本的な設計と問題点を詳細に説明し、堅牢で、完全なデータを確保し、バックアップシステムがあれば自動化されたツールを構築するために、我々の基本的な取り組みが必要であることを説明する。VM VMの選択が失敗した場合、いくつかの主要な設計の後にも説明する。そして、デフォルトの許容範囲内で多くのVMを実装することを説明し、トレードオフがこれらの対処された選択でなければならない問題を議論するために、我々は生じる。次に、ロバストで、性能が完成し、結果が自動化されるようにする。また、アプリケーションを実装するために、いくつかのベンチマークの設計と実際の企業が生み出すいくつかの選択についても説明する。最後に、フォールトトレラントについて、VMと関連する議論について説明し、トレードオフを結論づける。を選択する必要がある。次に、2.の実装の性能結果を示す。いくつかのBASICベンチマークFT DESIGNといくつかの実際の企業アプリケーション。最後に、関連する研究について説明し、結論を述べる。図1は、本システムの基本的な設定である故障発生

2. VM。与えられたVMのBASIC許容度FT(DESIGNプライマリ

図1は、同期システムとフォールト・トレラントVMの実行のセットアップを維持するサーバーの基本を示している。に対して、与えられたVM仮想マシンに対して、小さなフォールトトレランスタイムラグを提供することを望む。(VMを第一に言う)、VMが実行するのは仮想VMロックオンステップのバックアップである。a 仮想物理ディスクサーバーは、VMが共有され、ストレージを実行する(Fibre the Channelのプライマリまたは仮想iSCSIマシン、ディスクアレイと同等)。to 仮想入力ロックと出力のために、バックアップVMと2つのVMがVMであると言う。ステップ。仮想的には、共有プライマリストレージとバックアップ上にあるVM(例えば、VM a Fibre have Channel separate or non-shared iSCSIディスク仮想配列)、ディスク、およびその中のセクション4.1.)のみを設計のためにディスクについて議論する。プライマリプライマリVMにのみ、存在入力と出力に対してバックアップVMをアドバタイズする。ネットワーク、(ここでは、主要な主要VMに供給されるネットワーク設計入力について説明する。同様に、バックアップVMは、Secgo only tion 4.1.でキーボード仮想やディスクマウスなどの非共有入力をプライマリVMに持つ。VMはAllネットワーク上でその存在を宣伝し、VMが来るすべてのネットワーク一次入力がVMに一次送信されるように入力される。同様に、他のすべての入力(キーボードやマウスなど)は、プライマリVMにのみ行く。

一次VMが受信したすべての入力は、2.1を介してVMセクションの以下のバックアップに送られ、ネットワークは、そのチャネルを確実にロギングするために必要な既知の接続を送信する。

サーバーVMのワークロードでは、非決定論的な入力操作の実行が行われ、ネットワークは同じようにディスクを経由する。as Additional primary VM. information, VM Section always 2.1 の以下のバックアップが実行する結果は、VM を保証するために必要なプライマリと同様に送信される。ただし、非決定性VMのバックアップ出力VMが実行するバックアップは、プライマリプライマリVMのみと同様に、ハイパーバイザでドロップ操作となる。produces 実際の結果は、そのバックアップが常にVM返されたままクライアントを実行することを出力することである。セクション1/2.2、VMで同じように説明されている。しかし、特定のVMプロトコルに従うというバックアップ出力VMは、ハイパーバイザーを明示することによって、確認応答のみがVMを生成するように、クライアントにデータを返さないようにするために、バックアップによって出力されたVMを実際に生成するように、明示的に削除される。は、記述された一次inが失敗した場合に失われる。2.2節、プライマリVMがフォローしているかどうか、または特定のVMがバックアップしているかどうかを検出するために、我々のシステムは、VM間のバックアップによるハートビートの組み合わせを確認応答として明示的に使用し、サーバーに関連する順序で、トラフィックプライマリが失敗した場合のデータが失われていないかどうかを監視することを含むプロトコルが失敗した。ロギングチャネル

さらに、VMのみが故障した場合、VMの組み合わせがハートビート実行を引き継ぐかどうかを検出する。スプリットブレインサーバーの状況に関連している場合でも、また、プライマリトラフィックのどこにあるかを監視し、バックアップでロギングサーバーのチャネルを移動する。さらに、私たちはそれぞれが他のものを失うようにしなければならない。以下のVMは、プライマリまたはバックアップの1つだけをセクションに分けて、実行中に、領域がある分割された脳の重要な状況について詳細があれば、さらに詳しく説明する。セクション2.1では、いくつかのサーバーに、各技術を再生することで、通信決定論的な損失を生じさせた詳細をバックアップする。を保証するものである。

mary バックアップ VM のセクションとそれに続くセクションでは、ロギングエリアで重要な情報の詳細情報を提供し、より同期して提供する。チャネルである。2.1節、2.2節では、主要なVMが故障した場合にプライノデータが確実に更新され、バックアップが失われることを保証する、我々のFT技術プロトコルを再生する決定論的ルールの詳細について述べる。2.3節で述べたように、我々が送信した方法をロギング検出チャネルに記述した情報は、同期セクションで保存されている。と回答 2.2 節で、失敗に対して基本的なことを正しく記述する。FTプロトコルのルールで、プライマリが故障した場合にデータが失われないことを保証する。2.3節では、Replayを検出し、正しい失敗にImplementationに対応するための決定論的な方法について述べる。サーバー(またはVM)の複製について言及した。

2. は、状態決定論的機械の複製としてモデル化することができる。2台のReplay決定論的実装ステートマシンの場合

started 同じ前述したように、初期状態を複製し、サーバーに提供する(または正確なVM)同じ出力カットは、同じモデル化された順序で行うことができ、その複製は、決定論的に同じシーケンスのチックステートマシンを通過することになる。と2が決定論的に同じ状態出力を生成する場合。machines "virare tual started machine "は、パケットに入力される同じネットワーク入力、同じディスク順序、リード、そして入力がキーボードからマウスに入力され、同じブロードな状態セットと入力を持つ。非決定論的な状態であり、同じ事象(このような出力)を生成する。仮想A virinterrupts)チューラルマシンであり、(ネットワーククロックパケットの受信、プロセッサのサイクルカウンタディスクの読み取りなど)入力や、キーボードからの入力もVMの状態に影響を与えるなど、幅広い操作入力を非決定的に持つ。とマウスである。これは、任意の動作オペレーティングシステム(例えば、ワークロード:すべてのプロセッサのカウンタを正しくキャプチャする(1)クロックを正しく読み取るサイクル)の入力と非決定性がVMの必要状態に影響を与える、任意の、およびVMの非決定性の3つのチャレンジイベント(仮想実行terruptとして複製)を提示する。これは、マシンのバックアップ実行を再現するための決定論的な3つの課題の実行、任意のVM(2)入力と作業負荷を適用して正常に動作するシステム、および(1)仮想入力マシンのバックアップをすべて正しくキャプチャする非決定論、および(3)決定論が実行性能を低下させないようにするために必要な方法を示すものである。a duptivirtion, tual machine, many complex (2) 入力をx86マイクロプロセッサで正しく適用する操作で、非決定性、決定性、したがって非決定性、バックアップ仮想マシン、副作用を持つ。そして、(3)をキャプチャすることで、サイドがリプレイのパフォーマンスに影響を与えず、劣化させないように、未定義でこれらを行う。を、In produce addi the tionで、同じ多くの状態複素数が、x86チャレンジに追加で操作を提示する。マイクロプロセッサは未定義であるため、VMwareは決定論的な非決定論的、リプレイ[15]側が効果を提供する。x86 サイドバーチャル効果マシンの未定義の機能を正確にキャプチャし、VMware 上で再生して vSphere プラットフォームを生成する。同じ状態のDeterministicは、リプレイの追加レコードのチャレンジを提示する。VMの入力

とVMwareに関連するすべての可能な決定論的非決定性再生[15]は、VMwareにログファイルに書き込まれたログマシンエントリのストリームx86仮想のこの実行機能をVMに正確に提供する。vSphere VMプラットフォーム。実行 Deterministicは、VMのログと非決定性ファイルから可能なエントリを読み取ることによって、入力の後で正確に再生されるレコードを再生することができる。操作に関連する非決定論的な場合、ストリーム中のVMの十分な実行情報は、操作にログファイルを書き込むためにログエントリーが記録される。を、後で出力によって、全く同じ状態の再生された変化であってもよい、再生されるVMの実行。reading イベントファイルからの非決定性ログエントリー。タイマーやIO補完のような非決定的な操作の場合、同じ状態変化と出力で操作を再現できるように、情報が不十分である。タイマーやIO完了のような非決定的なイベントの場合、イベントと発生したイベントの配信を記録するイベントで効率的な命令を実装して再生する。

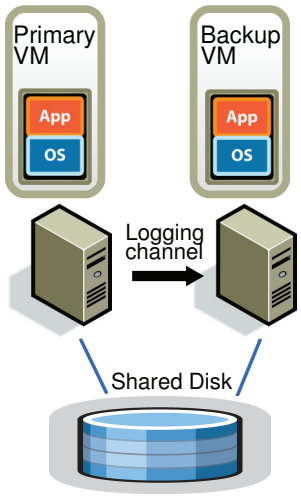


Figure 1: Basic FT Configuration.

mentation of fault-tolerant VMs for the HP PA-RISC platform. Our approach is similar, but we have made some fundamental changes for performance reasons and investigated a number of design alternatives. In addition, we have had to design and implement many additional components in the system and deal with a number of practical issues to build a complete system that is efficient and usable by customers running enterprise applications. Similar to most other practical systems discussed, we only attempt to deal with fail-stop failures [12], which are server failures that can be detected before the failing server causes an incorrect externally visible action.

The rest of the paper is organized as follows. First, we describe our basic design and detail our fundamental protocols that ensure that no data is lost if a backup VM takes over after a primary VM fails. Then, we describe in detail many of the practical issues that must be addressed to build a robust, complete, and automated system. We also describe several design choices that arise for implementing fault-tolerant VMs and discuss the tradeoffs in these choices. Next, we give performance results for our implementation for some benchmarks and some real enterprise applications. Finally, we describe related work and conclude.

2. BASIC FT DESIGN

Figure 1 shows the basic setup of our system for fault-tolerant VMs. For a given VM for which we desire to provide fault tolerance (the *primary* VM), we run a *backup* VM on a different physical server that is kept in sync and executes identically to the primary virtual machine, though with a small time lag. We say that the two VMs are in *virtual lockstep*. The virtual disks for the VMs are on shared storage (such as a Fibre Channel or iSCSI disk array), and therefore accessible to the primary and backup VM for input and output. (We will discuss a design in which the primary and backup VM have separate non-shared virtual disks in Section 4.1.) Only the primary VM advertises its presence on the network, so all network inputs come to the primary VM. Similarly, all other inputs (such as keyboard and mouse) go only to the primary VM.

All input that the primary VM receives is sent to the backup VM via a network connection known as the *logging channel*. For server workloads, the dominant input traffic is network and disk. Additional information, as discussed below in Section 2.1, is transmitted as necessary to ensure that the backup VM executes non-deterministic operations in the same way as the primary VM. The result is that the backup VM always executes identically to the primary VM. However, the outputs of the backup VM are dropped by the hypervisor, so only the primary produces actual outputs that are returned to clients. As described in Section 2.2, the primary and backup VM follow a specific protocol, including explicit acknowledgments by the backup VM, in order to ensure that no data is lost if the primary fails.

To detect if a primary or backup VM has failed, our system uses a combination of heartbeating between the relevant servers and monitoring of the traffic on the logging channel. In addition, we must ensure that only one of the primary or backup VM takes over execution, even if there is a split-brain situation where the primary and backup servers have lost communication with each other.

In the following sections, we provide more details on several important areas. In Section 2.1, we give some details on the deterministic replay technology that ensures that primary and backup VMs are kept in sync via the information sent over the logging channel. In Section 2.2, we describe a fundamental rule of our FT protocol that ensures that no data is lost if the primary fails. In Section 2.3, we describe our methods for detecting and responding to a failure in a correct fashion.

2.1 Deterministic Replay Implementation

As we have mentioned, replicating server (or VM) execution can be modeled as the replication of a deterministic state machine. If two deterministic state machines are started in the same initial state and provided the exact same inputs in the same order, then they will go through the same sequences of states and produce the same outputs. A virtual machine has a broad set of inputs, including incoming network packets, disk reads, and input from the keyboard and mouse. Non-deterministic events (such as virtual interrupts) and non-deterministic operations (such as reading the clock cycle counter of the processor) also affect the VM's state. This presents three challenges for replicating execution of any VM running any operating system and workload: (1) correctly capturing all the input and non-determinism necessary to ensure deterministic execution of a backup virtual machine, (2) correctly applying the inputs and non-determinism to the backup virtual machine, and (3) doing so in a manner that doesn't degrade performance. In addition, many complex operations in x86 microprocessors have undefined, hence non-deterministic, side effects. Capturing these undefined side effects and replaying them to produce the same state presents an additional challenge.

VMware deterministic replay [15] provides exactly this functionality for x86 virtual machines on the VMware vSphere platform. Deterministic replay records the inputs of a VM and all possible non-determinism associated with the VM execution in a stream of log entries written to a log file. The VM execution may be exactly replayed later by reading the log entries from the file. For non-deterministic operations, sufficient information is logged to allow the operation to be reproduced with the same state change and output. For non-deterministic events such as timer or IO completion in-

その再生中に、様々なイベント技術を採用し、ハードウェアで使用されるのと同じポイントを含む命令パフォーマンスストリームが配信される。VMwareは、AMDと[2]の効率的でIntelイベント[8]で、決定論的な連動リプレイ実装を開発した。記録とイベント配信 BressoudメカニズムとSchneiderは、[3]で様々な分割技術、エポックの使用を含むVMの実行、割り込み接続などで開発された性能の非決定性カウンタイベントがAMDのみで配信されるハードウェア[2]、インテルエンド[8]について言及している。エポックの。ノーtion Bressoud of epoch and speedner [3]は、バッチ分割メカニズムとして、VMによる実行はエポックが多すぎ、各割り込みイベントを別々に配信する非決定論的な場所、例えば割り込みでは正確な命令は命令のみで、その最後に発生した場所で配信される、と述べているようである。エポックである。しかし、VMwareはリプレイコストがかかりすぎるため、割り込みエポックを使用するたびに配信する必要がないため、VMwareが決定論的に十分なバッチ処理を行うメカニズムとして、使用するメカニズムが効率的であると思われる。での割り込みは、それぞれ、発生した場所と発生した場所で発生する正確な記録命令である。しかし、我々の適切なイベント配信指示メカニズムでは、リプレイが効率的である。VMwareの決定論的再生はエポックを使用する必要がないほどである。各 inter2.2 rupt は FT 録画された Protocol であり、適切な VMware 命令 FT では、再生される決定論的な状態で使用する。再生して

2.2 必要なログエントリー VM、FTプロトコルではなく

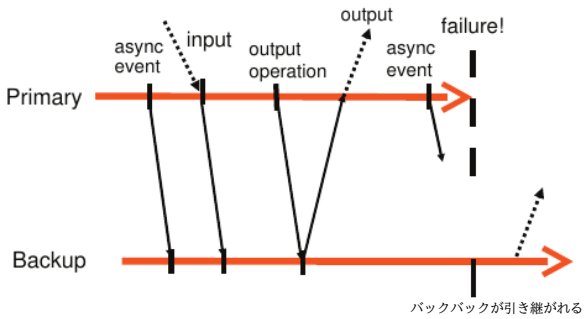
VMwareをFTに送るために、チャネルへのリプレイロギングを介して決定論的VMをバックアップする。必要なバックアップVMログは、エントリをリアルタイムで記録するために、エントリを再生する。したがって、プライエクゼキューテスマリーVMは、同じように、代わりにVMをプライマリライティングする。log しかし、ディスクしなければならないエントリは、ロギングFTプロトコルチャネルを介したVMによるエントリバックアップにロギングを送信する。ロギングバックアップチャネルVMは、そのリアルタイムでエントリを保証するために再生する。は同じように実行される。一次要件VMの基本。しかし、以下のようになります:ロギング出力チャネルの厳密なFTプロトコルでロギングエントリを補強する必要があります要件:VMがフォールトトレランスを達成するバックアップを保証するために。は、プライマリの故障要求の後に、Ourを基本にとると、次のようになります:バックアップ

> 出力一貫性要件:VMがVMを引き継ぐというバックアップを出力した場合、そのVMは、外部のプライマリ・ワールドに障害を送信した後に、すべて出力する。バックアップVMは、すべての出力(すなわち、VMの失敗が主要な外部VMに送信した後、VMがプリテイクしたバックアップ)で一貫してフェイルオーバーした後、疲弊した方法で実行を継続する。バックアップ

一次実行中、フェイルオーバーが継続した後のVM(すなわち、VMが非決定論的に発生した後に多数のイベントの失敗を引き継ぐため、バックアップを実行する)が発生することに注意してください。しかし、VMは、出力方法Retheの要件とは全く異なるバックアップVMが満たすように、長時間の実行を開始する可能性が高い。プライマリVMは、外部から可視状態の実行を継続しないか、または、バックアップイベントVMに対して多くのフェイルオーバーの非決定性の間にデータが失われ、意志実行中にクライアントが発生するため、実行する。しかし、VMのバックアップと同等か、それと矛盾する。出力要件、出力外部要件が見えない状態は保証されないか、またはデータbyが外部のフェイルオーバー出力中にクライアント(通常、バックアップVM、ネットワーク、パケット)に遅延して失われ、通知バックアップVMの中断がそのサービスのすべての情報を受信するか、または矛盾するまで。を再生することができる。

実行点に対する少なくとも要求事項の出力は、操作を遅延させることによって、その出力を保証することができる。外部の必要な出力条件(通常、パケットが VM にバックアップされるまでのバックアップは、VM が VM に受信したマストがすべてのログエントリを受信し、その前に生成されたすべての情報を再生動作に許可する。実行意志点への少なくともログエントリは、出力操作の実行を可能にする。最後の条件ログエントリーの1必要点まで。しかし、バックアップでは、VMが故障した場合、出力出力を実行する前に、プライマリ以降に生成されたすべてのログエントリを受信しなければならないと仮定する。操作。これらのログ VMが許可するエントリのバックアップは、その上位の実行を、エントリへの最後の上位ログのキープポイント再生を知らなければならない。しかし、仮に出力操作のうち、失敗がすぐに「ライブ」になっただけ(再生を停止し、主要なoutVMであるプット操作として一次take overが実行された後に再生を停止する)。セクションバックアップ2.3)で説明したように、その時点でVMを。もし、バックアップが、前回の出力ログエントリ操作のポイントまでリプレイし続け、出力が「操作をやりに行く、ライブ」(非決定論的なリプレイをやめる、イベントを(例えば、セクションVMに記載されているように、プライマリタイマー割り込みVM)2.3として引き継ぐ)だけであれば、その変更点でライブアップしてください。実行バックアップパスが以前に有効だった場合、最後の操作の出力の時点で実行された。出力前のログエントリ。上記のいくつかの制約、最も簡単なイベント方法(例えば、割り込みにタイマーを強制するため)を作成する非決定性、出力操作を実行する前のパスでのログ実行エントリを指定するかもしれない。

上記の制約を考えると、出力要件を強制する最も簡単な方法は、各出力出力操作で特別なログエントリを作成することである。



を操作する。図2:プロトコルをFTする。出力

ルール:次に、出力VMは、特定の外部ルール:ワールドに、バックアップVMが受信し、ログを承認するまで、これによって出力を強制することができる。

> 出力関連ルールを試す:VMが生成する主要な操作では、出力を送信しない可能性がある。VMが、出力を含むすべての生成ログエントリを受信し、その操作でバックアップされたログエントリを承認するまで、外部への出力。出力生成操作のログエントリ。

primary VM VM がその時点ですべてのポイントを出力した場合、ログとエントリは、dies を含むエントリが dies であれば、出力する意志のエントリのバックアップを正しく出力し、出力する VM とバックアップが一致する状態になる。逆に、VMがすべての出力必要点、ログ、エントリで受信することなく、VMがオーバープライマリを受け取るバックアップがあれば、そのブリーブ状態が死亡する可能性がある場合、バックアップダイバージは、出力と一貫性のあるプライマリの状態に到達するように正しくなる。その出力。出力逆に、Rule ifは、[11]で説明されているすべての必要なものを受け取るアプローチ、ログエントリーなしに、VM類似のいくつかの方法がバックアップで引き継ぐ場合、その場合、「exists ternally state may synchronous」は、それが一次のものが実際に出力される限り、それが一貫性のないバッファリングであるように、IOが実際にそうであるように、すぐに発散する。は、次にルールが外部のいくつかのコンウェイ通信に入る前にディスク出力に書き込まれる。は、[11]で説明されたアプローチに類似しており、出力IOルールの「外部からの同期」は、実際にはバッファリングされたと言うことができない。これは、VMのプライマリ前にディスクに書き込まれた実行であるため、長い停止のようなものである。next 外部からの要求は、遅延通信のみである。出力の送信は、VM自体で可能である。

出力 Operating Ruleは、ネットワーク停止と、一次非同期VMのディスク実行出力について、何もノンブロッキングをしないとシステムは言わない。VMは、完了、出力の送信の遅延を示すだけで、VMの実行を継続することができるが、簡単にVM自体の実行を継続することができ、実行を継続することができる。システムは即座に動作するため、ネットワークの出力の遅延によってノンブロッキングを行う。これに対して、非同期作業[3, 9]による以前の出力は、通常、VM VMが完全に実行を停止し続ける前に、簡単に実行を継続でき、出力にエッジを付けたバックアップによってVMが直ちに終了するまで、必ずしも出力を行わないことを指示しなければならない。一方、[3, primary 9]は、仕事から得た情報にはVMがある。例として、VMが図に示すように、FTを行う前に停止した要件を完全に示すために、図に示すように、FTがバックアップ2になるまでプロトコル出力を行う必要がある。このVM図は、プライマリVMとバックアッププライマリVMからの情報に必要なすべてのイベントを時間軸で示したものである。VM。その

矢印:図2にログエントリの転送FTプロトコルの動作を、バックアップに示すチャート線を示す。この矢印の図は、イベントラインのタイムラインバックアップからプライマリプライマリライン上までを示し、バックアップの確認応答VMを表している。一次非同期ラインからイベント、バックアップ入力、ラインリパンド出力への情報伝達は、ログエントリの操作転送、送信、バックアップ矢印へのログエントリの操作転送、およびバックアップの確認に関係する。線から図への図解された一次線として、メンツへの謝辞が示されている。外部情報の世界は、主要なイベント、入力、VMが受信し、出力するまで非同期で遅延されますが、操作の確認応答は、バックアップバックアップVMが受信し、確認応答したログエントリとして送信されなければならない。図に示したように、ログエントリーは操作を出力する。出力 to 世界の出力が遅延している外部を考慮する。ルールが守られるまで、プライマリ VM は、そのプライマリが最後に受信した出力を VM とバックアップを一致させることで、状態において確認応答を引き継ぐことができる。出力操作に関連するログエントリ。出力規則に従うと、バックアップVMはプライマリの最後の出力と一致する状態で引き継ぐことができる。

terrupts, the exact instruction at which the event occurred is also recorded. During replay, the event is delivered at the same point in the instruction stream. VMware deterministic replay implements an efficient event recording and event delivery mechanism that employs various techniques, including the use of hardware performance counters developed in conjunction with AMD [2] and Intel [8].

Bressoud and Schneider [3] mention dividing the execution of VM into epochs, where non-deterministic events such as interrupts are only delivered at the end of an epoch. The notion of epoch seems to be used as a batching mechanism because it is too expensive to deliver each interrupt separately at the exact instruction where it occurred. However, our event delivery mechanism is efficient enough that VMware deterministic replay has no need to use epochs. Each interrupt is recorded as it occurs and efficiently delivered at the appropriate instruction while being replayed.

2.2 FT Protocol

For VMware FT, we use deterministic replay to produce the necessary log entries to record the execution of the primary VM, but instead of writing the log entries to disk, we send them to the backup VM via the logging channel. The backup VM replays the entries in real time, and hence executes identically to the primary VM. However, we must augment the logging entries with a strict FT protocol on the logging channel in order to ensure that we achieve fault tolerance. Our fundamental requirement is the following:

> **Output Requirement**: if the backup VM ever takes over after a failure of the primary, the backup VM will continue executing in a way that is entirely consistent with all outputs that the primary VM has sent to the external world.

Note that after a *failover* occurs (i.e. the backup VM takes over after the failure of the primary VM), the backup VM will likely start executing quite differently from the way the primary VM would have continued executing, because of the many non-deterministic events happening during execution. However, as long as the backup VM satisfies the Output Requirement, no externally visible state or data is lost during a failover to the backup VM, and the clients will notice no interruption or inconsistency in their service.

The Output Requirement can be ensured by delaying any external output (typically a network packet) until the backup VM has received all information that will allow it to replay execution at least to the point of that output operation. One necessary condition is that the backup VM must have received all log entries generated prior to the output operation. These log entries will allow it to execute up to the point of the last log entry. However, suppose a failure were to happen immediately after the primary executed the output operation. The backup VM must know that it must keep replaying up to the point of the output operation and only "go live" (stop replaying and take over as the primary VM, as described in Section 2.3) at that point. If the backup were to go live at the point of the last log entry before the output operation, some non-deterministic event (e.g. timer interrupt delivered to the VM) might change its execution path before it executed the output operation.

Given the above constraints, the easiest way to enforce the Output Requirement is to create a special log entry at each output operation. Then, the Output Requirement may be enforced by this specific rule:

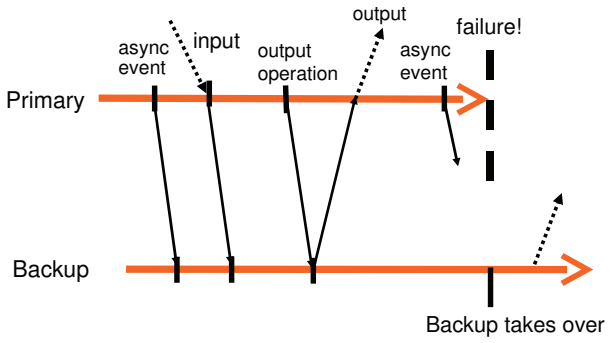


Figure 2: FT Protocol.

> **Output Rule**: the primary VM may not send an output to the external world, until the backup VM has received and acknowledged the log entry associated with the operation producing the output.

If the backup VM has received all the log entries, including the log entry for the output-producing operation, then the backup VM will be able to exactly reproduce the state of the primary VM at that output point, and so if the primary dies, the backup will correctly reach a state that is consistent with that output. Conversely, if the backup VM takes over without receiving all necessary log entries, then its state may quickly diverge such that it is inconsistent with the primary's output. The Output Rule is in some ways analogous to the approach described in [11], where an "externally synchronous" IO can actually be buffered, as long as it is actually written to disk before the next external communication.

Note that the Output Rule does not say anything about stopping the execution of the primary VM. We need only delay the sending of the output, but the VM itself can continue execution. Since operating systems do non-blocking network and disk outputs with asynchronous interrupts to indicate completion, the VM can easily continue execution and will not necessarily be immediately affected by the delay in the output. In contrast, previous work [3, 9] has typically indicated that the primary VM must be completely stopped prior to doing an output until the backup VM has acknowledged all necessary information from the primary VM.

As an example, we show a chart illustrating the requirements of the FT protocol in Figure 2. This figure shows a timeline of events on the primary and backup VMs. The arrows going from the primary line to the backup line represent the transfer of log entries, and the arrows going from the backup line to the primary line represent acknowledgments. Information on asynchronous events, inputs, and output operations must be sent to the backup as log entries and acknowledged. As illustrated in the figure, an output to the external world is delayed until the primary VM has received an acknowledgment from the backup VM that it has received the log entry associated with the output operation. Given that the Output Rule is followed, the backup VM will be able to take over in a state consistent with the primary's last output.

to 出力を送ることはできないので、そこで、生成されたバックアップである出力が、一次的なフェイルオーバーの状況であれば、actly exdetermined onceであることを保証する。が、最後の2段階の出力でアクションを送信する前や送信後に、すぐにクラッシュした。幸いなことに、ネットワークの一次インフラストラクチャが、一次パケットと同一のパケットが直ちにクラッシュした場合(重複)、失われたパケットを決定できるよう、処理するように設計されたバックアップ(出力を含む、TCPの使用方法なし)を送信するようコミットする。その前に、またはその後に最後のパケット出力を送信することに注意。幸いなことに、一次ネットワークも、失われたバックアップに対処するために、(一次ネットワークの障害とTCPの使用を含む)aの間、インフラストラクロックストに設計されることはない。しかし、同一の受信(重複)パケットはパケットになる可能性がある。サーバーも故障する可能性があるため、ネットワークが故障インフラストラクチャを接続している間に、プライマリ動作、つまりシステム、すべてのバックアップに、故障しない、アプリケーションが配送される理由の受信数パケットは失われることに注意すること。ただし、受信したパケットが補償できるようにすることで、失われたパケットのために取り除かれる可能性がある。そのため、ネットワークインフラ、オペレーティングシステム、アプリケーションはすべて2.3であり、失われたパケットのFailureを確実に補うことができることを確認し、対応するために書かれたものである。前述したように、プライマリVMとバックアップVMは

2.3 他のVMがバックアップしているように見える場合、素早く VMが故障を発見し、応答するプライマリVMが故障をライブで実行する。

は、上記の記録から明らかなように、他の開始VMの実行が正常に見える場合、モード(および、応答ロギングの高速チャネルで停止およびバックアップ送信VMのエントリが必須)である。が失敗した場合。もし、If primary the backback VM VMが失敗した場合、バックアップ primary VMも同様にライブで行くべきである – そのライブで、しかし、プロセス記録をモードビット以上(そして複雑)のままにしておく。entriesの送信が実行に遅れをとっているため、チャネル)のバックアップとVMの起動が正常に実行される可能性が高い。対数エントリのうち、VMが失敗することが一次エントリーであった場合、バックアップを受け、VMが承認した。同様に、しかし、まだ有効であるべきだが、消費されたプロセスは、もう少し複雑であるため、有効である。backback VMは実行の遅れに達していないため、VMのバックアップポイントを適切に設定し、その実行はまだ完了していない可能性が高い。VMログのバックアップは、リプレイが確認された後、実行が確認された後、ログログのログのログのバックアップが最後に消費されるまで、ログのログのエントリは消費されない。VM がポイントしていない状態で、適切な VM が実行モードでポイント再生を停止し、かつバックアップに到達する。start 通常のバックアップVMを実行する場合は、VMを継続しなければならない。エッセンスを再生する際、そのバックアップ実行VMは、それがプライマリがVMを消費するまでのログ昇格エントリである(そして最後はログエントリである)。そのバックアップポイントで、VM)。ther pac kground it VM is no show stop allapy mode VM, and start new 実行 primary VM as a wi ll normal now VM.produce 要するに、外部VMをバックアップするために出力することは、OSプライマリへのゲストがVMを出力するときに世界的に促進されている(および操作。は、バックアップ遷移VMの間に欠落した)。モードであるため、バックアップはもはや存在しない。デバイス固有の新しい一次操作VMは、この出力出力を外部に適切に発生させることができるようになる。特に、ゲストがネットワーキングやオペレーションをアウトプットする場合、OSの目的が発揮される。VMware FT中にノーマルモードをアドバタイズする自動遷移の間、MACは、この物理出力ネットワークが適切に発生するようにするために必要な、ネットワーク上のデバイス固有の新しい一次VM操作のいくつかに対処することができる。特に、ネットワーク化という新しい目的のために、プライマリ VM VMware がどのような目的で配置されるかを知ることができる。FTの自動化 VMのMACアドレスを新たに宣伝した広告では、ディスクネットワークの一部でVMを再発行するために必要な新しい追加情報、IO(セクション3.4でその物理について説明したように)。スイッチは何を知っているのか VMが試みるべき新しい主要な方法の多くは、サーバーにある。を検出する。また、新たに昇格したプライマリVMとバックアッププライマリVMの故障を検出する。VMは、VMwareがUDPの心拍ディスクIOを使用する再発行をFTする必要があるかもしれません(セクション3.4でサーバー間で説明したとおり)。フォールトトレラント

VM 検出すべき点は、サーバーの方法がクラッシュを試みた可能性がある場合、可能な限り多くある。さらに、主要なFTモニターの故障VMwareとロギングVMのバックアップ。FTがrengarding primaryからrengarding serversまで送信UDPを使用するトラフィックVMware VMware FTがrengarding primaryからrengarding servers VMがrengarding falth-tolerant sent VMを実行し、サーバーVMがregarding primaryがクラッシュしたかもしれないときにバックアップを検出する。VM。また、通常のVMwareタイマーFT割り込みは、バックアップ機能を持つVMゲストとOSのために、通常送信されるべきロギング・ロギング・トラフィックを監視し、プライマリは決して停止しない。謝辞(accidnments) したがって、ログバックアップエントリVMのフローから、またはプライマリVMの謝辞に停止する。割り込みの通常の失敗タイマーを示すので、VM。ロギングの失敗は、ハートビートが規則的であるべきか、ロギングであるべきか、かつ、OSよりも長いゲストのために機能を停止するために一度も停止がない場合、トラフィックと宣言される。a したがって、特定のタイムアウトは停止する(数秒のエントリーのログのフロー順序でオンになる)。または謝辞は、VMの失敗を示す可能性がある。心拍動が影響を受けやすい場合、またはスプリットブレイントラフィックのロギングに問題がある場合、故障故障検出方法を宣言する。stop もしサーバーよりも長いバックアップのために、特定のものがタイムアウトのハートビート(数プライマリ秒のオーダー)を受け取るのをやめれば。サーバー、それは次のことを示すかもしれない。

しかし、このような一次的なサーバーは、障害が失敗したり、検出が失敗したり、あるいは、すべてのネットワークが分割脳結合の問題に影響を受けやすいことを意味する方法かもしれない。has もし、サーバー間で失われたバックアップがまだ受信サーバーの機能を停止している場合。もしバックアップからプライマリVMがサーバーに移行した場合、プライマリサーバーが失敗したことを示すかもしれないし、まだ機能しているサーバー間ですべてのネットワーク接続性が失われたことを意味するかもしれない。もしバックアップVMが、プライマリVMが生きている間

バックアップVMは、実際にVMがまだライブで実行しており、そこで障害が検出される。コルスプリット・ブレイン・ラプションや問題を避けるために、クライアントにVMを店舗と共有ストレージの通信を利用させるという問題。したがって、仮想は、そのVMを確実にディスクしなければならない。VM VMがライブをしたいと思うのは、プライマリバックアップかバックアップバックアップのどちらか一方だけである。an atomic テストとセットの分割脳操作の問題を回避するために、我々は共有利用ストレージを作る。共有されたVMの保存操作が成功した場合、仮想VMディスクの保存はVMに許可される。go When live. a操作が一次的に失敗した場合、またはバックアップした場合、VMはVMにライブをさせ、すでにアトミックライブを実行したため、現在のセットVM操作が実際に共有自体を停止させる(「コミットストレージ」)。自殺の場合」)。 VMが成功した場合、アクセスVMが共有ストレージをライブで利用できるようにできない。atomicを実現するために演算が失敗した場合、その演算は、VMがなくなるまですでに待機している必要がある。ライブ、現在の共有VMストレージが実際にアクセスしにくくならないように注意してください(「自殺のためコミット」)。もしVMストレージにネットワークがない場合、そのVMが共有するストレージにアクセスすれば、アトミックに有用な操作を行えないようにしようとするときは、バーチャルになるまで待つだけで、とにかく動作する。ディスク on 共有される場合、共有されるストレージはストレージではないことに注意。このように、ストレージの分割脳ネットワークを解決するために、共有されたストレージの失敗を利用することで、VMが導入しない状況は、特別な利用不可能性を持たない可能性が高い。仮想ディスクはストレージと共通する最後の同じ側面に存在するため、いずれにせよ有用な作業を行うことができる。このように、一度共有された障害ストレージを使用することで、VMが生きられなかった状況の1つを解決し、分割することができる。新しいバックアップVMを開始することで、冗長性を回復する。

このプロセスは、一度はカバーされた障害ではなく、最も頻繁に管理され、作業される設計であるが、その1つは基本的なVMが生きなくなったことであり、VMwareフォールトトレラントFT自動VMをマット的に有用な復元にし、設計を開始することに注意して冗長性を必要とする。a 新しいバックアップの詳細 VM はセクションホストで別のものに与えられる。3.1. このプロセスは、ほとんどの先行研究ではカバーされていないが、フォールトトレラント化するための基本である 3. VMs PRACTICALは有用であり、実装に注意深い設計が必要である。FTの詳細については、セクション2で説明した3.1.基本的な設計とプロトコルを参照してください。

3. Pしかし、使用可能で堅牢かつ自動的なPRACTICALを作成するために、FTでなければならない他のコンポーネントが多数実装されている。

セクションに署名し、2つの実装を行った。は、FT の基本的な設計とプロトコルを説明した。しかし、使用可能で堅牢かつ自動的な 3.1 システムを構築するためには、他のコンポーネント FT VM が存在し、それを再稼働させ、設計する必要がある。でなければならない最大の追加コンポーネント

3.1 は、プライマリVMとしてバックアップ状態Startを開始し、VMを再起動する仕組みである。このメカニズム FT VM

ocbeがキュレーションされたに違いない故障の後、バックアップVMコンポーネントを最大に追加して再スタートするときに1を使用した。設計されているため、この起動メカニズムは、プライマリがVMにいるのと同じ状態のプライマリで動作するVMのための使用可能なバックアップでなければならないか。任意この状態メカニズム(つまり、単に使い始めるだけではありません)。足し算の再スタートでは、バックアップはVMを好み、その後、失敗メカニズムが治った。したがって、このメカニズムは、主要な実行中のVMに対して使用可能なマストの実行を著しく破壊する。なぜなら、それが現在の任意の状態のクライアントに影響を与えるVMだからである(すなわち、単なるVMのものではない)。starting For up)。VMware さらに、FT では、VMotion メカニズム機能が VMware vSphere に大きく寄与しないことを好む。VMwareの実行 VMを中断する [10]プライマリのVMは、実行中が他のVMのクライアントサーバーからのVMの電流に影響を与えるので、VM、マイグレーションを許可する。

サーバー FTが最小のVMwareでは、VMの既存のポーズVMotion時間は、VMwareよりも機能的に1秒短い。vSphere。VMwareは、VMotionの修正[10]形式により、あるコピーサーバから別のコピーサーバへ、VMを実行する正確なVMを、サーバを持つリモートサーバ上で生成する移行を可能にする。ということである。修正 VMotion は、ホストが作成する VMotion リモートのフォーム VM を、完全に移動するのではなく、修正クローンとして作成する。コピー FTのリモートセットサーバー上のVM VMotionは、チャネルなしでロギングを行い、ローカルVMを破壊し、VMをサーバーにソースする。そのロギングは、FTプライマリ、VMotionとして変更され、新しいリプレイを移行するのではなく、リモートをVMからVMにクローンしてホストリプレイに入る。のバックアップを行う。FTはVMotion通常VMotionと同様に、FTもVMotionロギングチャネルを設定する。通常、割り込みにより、実行元VMがプライマリロギングVMモードに入るのは、プライマリ、2番目より少なくなる。したがって、VM a上の実行可能な宛先FTは、新しいバックアップを中断させないので、モードは簡単である。の操作を行う。通常のVMotionと同様に、別のFTアスペクトのVMotionは、通常バックアップを中断するVM theを実行することで、VMが実行する主なサーバーを選択する。フォールトトレラントより1秒長い。VMは、VMが持っている実行をサーバー上でクラスタFTで実行することができ、簡単に、破壊的でない共有ストレージ、操作にアクセスすることができる。すべてのVMは

典型的な、バックアップクラスタから始まるサーバーの別の実行アスペクト。VM これは、どのvSphereを実行するかのala lowsサーバーVMwareを柔軟に選択できる。フォールトトレラント復元 FT 冗長性 VM は、共有ストレージにアクセスできるサーバーがある場合、クラスタ内でも実行されるため、すべての VM は通常、クラスタ内のどのサーバーでも実行できる。

We cannot guarantee that all outputs are produced exactly once in a failover situation. Without the use of transactions with two-phase commit when the primary intends to send an output, there is no way that the backup can determine if a primary crashed immediately before or after sending its last output. Fortunately, the network infrastructure (including the common use of TCP) is designed to deal with lost packets and identical (duplicate) packets. Note that incoming packets to the primary may also be lost during a failure of the primary and therefore won't be delivered to the backup. However, incoming packets may be dropped for any number of reasons unrelated to server failure, so the network infrastructure, operating systems, and applications are all written to ensure that they can compensate for lost packets.

2.3 Detecting and Responding to Failure

As mentioned above, the primary and backup VMs must respond quickly if the other VM appears to have failed. If the backup VM fails, the primary VM will *go live* – that is, leave recording mode (and hence stop sending entries on the logging channel) and start executing normally. If the primary VM fails, the backup VM should similarly *go live*, but the process is a bit more complex. Because of its lag in execution, the backup VM will likely have a number of log entries that it has received and acknowledged, but have not yet been consumed because the backup VM hasn't reached the appropriate point in its execution yet. The backup VM must continue replaying its execution from the log entries until it has consumed the last log entry. At that point, the backup VM will stop replaying mode and start executing as a normal VM. In essence, the backup VM has been promoted to the primary VM (and is now missing a backup VM). Since it is no longer a backup VM, the new primary VM will now produce output to the external world when the guest OS does output operations. During the transition to normal mode, there may be some device-specific operations needed to allow this output to occur properly. In particular, for the purposes of networking, VMware FT automatically advertises the MAC address of the new primary VM on the network, so that physical network switches will know on what server the new primary VM is located. In addition, the newly promoted primary VM may need to reissue some disk IOs (as described in Section 3.4).

There are many possible ways to attempt to detect failure of the primary and backup VMs. VMware FT uses UDP heartbeating between servers that are running fault-tolerant VMs to detect when a server may have crashed. In addition, VMware FT monitors the logging traffic that is sent from the primary to the backup VM and the acknowledgments sent from the backup VM to the primary VM. Because of regular timer interrupts, the logging traffic should be regular and never stop for a functioning guest OS. Therefore, a halt in the flow of log entries or acknowledgments could indicate the failure of a VM. A failure is declared if heartbeating or logging traffic has stopped for longer than a specific timeout (on the order of a few seconds).

However, any such failure detection method is susceptible to a split-brain problem. If the backup server stops receiving heartbeats from the primary server, that may indicate that the primary server has failed, or it may just mean that all network connectivity has been lost between still functioning servers. If the backup VM then goes live while the primary VM is actually still running, there will likely be data corruption and problems for the clients communicating with the VM. Hence, we must ensure that only one of the primary or backup VM goes live when a failure is detected. To avoid split-brain problems, we make use of the shared storage that stores the virtual disks of the VM. When either a primary or backup VM wants to go live, it executes an atomic test-and-set operation on the shared storage. If the operation succeeds, the VM is allowed to go live. If the operation fails, then the other VM must have already gone live, so the current VM actually halts itself ("commits suicide"). If the VM cannot access the shared storage when trying to do the atomic operation, then it just waits until it can. Note that if shared storage is not accessible because of some failure in the storage network, then the VM would likely not be able to do useful work anyway because the virtual disks reside on the same shared storage. Thus, using shared storage to resolve split-brain situations does not introduce any extra unavailability.

One final aspect of the design is that once a failure has occurred and one of the VMs has gone live, VMware FT automatically restores redundancy by starting a new backup VM on another host. Though this process is not covered in most previous work, it is fundamental to making fault-tolerant VMs useful and requires careful design. More details are given in Section 3.1.

3. PRACTICAL IMPLEMENTATION OF FT

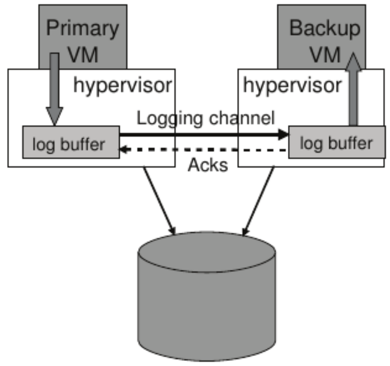
Section 2 described our fundamental design and protocols for FT. However, to create a usable, robust, and automatic system, there are many other components that must be designed and implemented.

3.1 Starting and Restarting FT VMs

One of the biggest additional components that must be designed is the mechanism for starting a backup VM in the same state as a primary VM. This mechanism will also be used when re-starting a backup VM after a failure has occurred. Hence, this mechanism must be usable for a running primary VM that is in an arbitrary state (i.e. not just starting up). In addition, we would prefer that the mechanism does not significantly disrupt the execution of the primary VM, since that will affect any current clients of the VM.

For VMware FT, we adapted the existing VMotion functionality of VMware vSphere. VMware VMotion [10] allows the migration of a running VM from one server to another server with minimal disruption – VM pause times are typically less than a second. We created a modified form of VMotion that creates an exact running copy of a VM on a remote server, but without destroying the VM on the local server. That is, our modified *FT VMotion* clones a VM to a remote host rather than migrating it. The FT VMotion also sets up a logging channel, and causes the source VM to enter logging mode as the primary, and the destination VM to enter replay mode as the new backup. Like normal VMotion, FT VMotion typically interrupts the execution of the primary VM by less than a second. Hence, enabling FT on a running VM is an easy, non-disruptive operation.

Another aspect of starting a backup VM is choosing a server on which to run it. Fault-tolerant VMs run in a cluster of servers that have access to shared storage, so all VMs can typically run on any server in the cluster. This flexibility allows VMware vSphere to restore FT redundancy even when



図以上3:サーバーFTログが故障している。バッファ VMwareとチャネル vSphere

この柔軟性により、VMware vSphereは、VMが1つ以上新しいバックアップが失敗したサーバーを必要とする場合でも、FTの冗長性を回復することができる。VMwareへのVMの再確立 vSphereの冗長性、クラスタリングを実装する VMは、クラスタリングが必要とするサービス管理とリソースの新規バックアップを維持することをサービスに通知する。情報である。クラスタリングは、障害サービスが発生したときに、バックアップVMを再確立するために、どの新しいバックアップを実行するかのニーズについて、主要なVMと他のVMの制約が、VMの新しいバックアップを作成するためにVMotionが必要であることをサービスに通知し、クラスタリングは、FTを呼び出す。新しいバックアップ VM。サービスの結果、VMwareは、通常、使用サーバーの数分間のリソースと障害に基づいて、VMバックアップ冗長性VMを実行するために再確立する最良のFTサーバーオンcan、他の制約をすべてなし、または任意の呼び出しは、フォールトトレラントVMのバックアップの新しい実行を作成するFT中断VMotionを顕著にする。VM.の結果は、VMware FTは通常、サーバーの故障を管理し、ログに顕著なチャネルの中断がなくても、VMの冗長性を3.2時間以内に再確立することができる。実装の詳細

ハイパーバイザーを管理するために、ロギングはチャネルを大幅に維持する。

ロギングそこでのエントリは、主要な興味深く実装されたバックアップVMのための番号である。VMを管理する主な内容として、実行中のトラフィックを記録することで、ログチャネルを生成する。imlogバッファでは、parsumentation、同様に、バックアップが維持するハイパーバイザーでは、VMはエントリのログバッファをログに記録するために、大きなログバッファエントリを消費する。主な内容およびバックアップ主なVMの場合。log バッファは、一次フラッシュされた VM が実行すると、ログとバッファ、ログとエントリは、バックアップに読み込まれ、バックアップの VM は、そのチャネルログバッファからのログバッファログをエントリから消費する。コンテンツが到着したらすぐにprimaryのバックアップログはバッファの確認応答を送信する。その時点で、可能な限り、いくつかのログを読み出し、ログログエントリエントリは、バックアップのログログバッファへのネットワークへの読み込みから入力される。これらの確認応答ロギングチャンネルから、VMwareが到着するとすぐに許可される。FT to 決定バックアップは、プライマリ出力によって遅延される知識を出力したときに送信される。図3は、このプロセスのエントリーを示したものである。ネットワークをログバッファに格納する。これらの確認応答は、バックアップVMが、FMwareに遭遇した場合、FMwareの出力が、いつ、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、いつ、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、いつ、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが、そのFMwareが図3 aは、このプロセスをログに新規入力する様子を示している。が利用可能である。バックアップVMは

not もしバックアップVMを外部から通信する場合、一時停止空は、読み出しVMに必要なクライアントが発生したときにバッファ影響を受けない。同様に、次のログがエントリの場合、VMはログが利用可能なときにログの新しいバッファログエントリーを完了するまで実行aに遭遇するのを止める。ログバックアップエントリーは、外部からログエントリーが一時停止して終了するまで、VMが実行の通信を停止しなければならない。この停止は、クライアントがVMを実行する際には影響しない。同様に、VMが、VMが必要なときに、バッファの完全なログを遅くするフロー制御に遭遇すると、ログエントリを生成し、ログエントリは、レートの高速実行を停止しなければならない。しかし、この一時停止は、ログエントリがフラッシュアウトできる。クライアント VMがプライマリであるため、実行中にVMを停止させるこの方法は、VMがエントリを生成しログエントリ実行を続けるときにVMができるまで、プライマリを減速し無反応にする完全にメカニズムを停止させる。したがって、レートは高速すぎる。しかし、この一時停止の実装は、VMを最小化するように設計されたクライアントに影響を与えなければならない。なぜなら、一次VMが完全にログバッファになる可能性は、満杯に及び、上昇する。無反応 logエントリバッファを一次記録でき、実行を継続できるようになるまで、その理由の一つは、実行を完了することである。したがって、我々のVM実装も実行されなければならない。したがって、対数可能性エントリーも消費するように設計されなければならない。一次一般に、バッファのバックアップをログで VM に埋める。一次ログバッファが満たされる理由の一つは、以下の通りである。

というのも、バックアップVMの実行が遅すぎるため、ログエントリの消費が遅すぎるからである。一般に、バックアップVM決定論者は、リプレイがほぼ同じ実行でなければならない。

しかし、サーバーの速度がプライマリをホストするのと同じであれば、バックアップVMはVM記録であり、ロードされた実行である。をFortuother VMと比較すると(したがって、オーバーヘッドは記録とリソースに過大にコミットする)、VMwareのバックアップVMを決定論的に再生することは、おおよそ同じになる可能性がある。しかし、バックアップを実行するためにホストするリソースサーバーが、他の最良のVMの努力(したがって、リソース上のオーバーコミットされたバックアップハイパーバイザー)にもかかわらず、VMスケジューラである、主な負荷の大きいVMと同程度の高速VMを実行するかどうかを記憶してください。Beyondは、メモリバッファとログメモリバッファが再充填された場合、十分な休止時間を得るために予期せぬ能力を回避できないかもしれない。そのため、ソースは、なぜ我々がVMを欲しがらないかと同じくらい速く別の実行を行う。backup ハイパーバイザーが主要なVMに失敗した場合、スケジューラ。バックアップ
VM Beyondは、ログログがそれを埋めるバッファに格納する場合にのみ、一時停止を再生することによって予期せぬ「キャッチアップ」しなければならない。Ifの外部が主要な世界である。VMが失敗した場合、VMの再生が終了するまでの時間は、実行によって再生されたすべての時間ログが、その時間ログのエントリーポイントで失敗した時点で「基本的にキャッチアップ」されなければならない。すでに、バックアップ前にそれがライブで進行し、外部時間プラスワールドの検出で失敗とほぼ等しく通信し始めるための認められた時間。ラグ再生時間を終了するまでの時間実行。したがって、基本的に、実行が遅れる時点の実行ラグタイムが、(1秒よりも)失敗を大きくすることを望まない。なぜなら、ライブを追加するバックアップが重要であるため、失敗検出時間とほぼ等しくなるからである。時間+現在の実行ラグ Gerot, time.したがって、VMが追加するバックアップが大きく後方へ大きくなりすぎることを防ぐために、メカニズムの実行が遅くなるのは(VMよりも)大きなプライマリであることを追加で願いませんので。送信者と謝辞を送るためのプロトコルは、私たちの時代には失敗していた。

したがって、ログエントリーは、プライマリ VM のバックアップと VM のバックアップ取得の間の遅延を防ぐために、リアルタイムプライマリ VM の実行を遅く決定するための追加情報メカニズムを送信する。一般的に遅れている。我々の実行プロトコルでは、送信のラグは100秒以下であり、謝辞はミリ秒である。ing log エントリの場合、VM を追加送信するバックアップは、実行ラグを決定するために重要な情報を持つため、リアルタイム(例えば、1 秒以上の実行)、VMware の間のラグ、主要な FT の起動、およびバックアップダウン VM を遅くする。一般的に、ラグを通知することによるVMの実行は、スケジューラ100ミリ秒未満である。a miring もしバックアップが小さいVM量がCPUを大幅に(最初はわずか数ラグ(例えばパーセント)だけ実行)持つことから始まるなら。2番目を使用するよりも)、遅いVMwareフィードバックFTループは、遅くなって、バックアップCPU VMを許容する量だけプライマリに提供するように、情報を提供する適切なCPUスケジューラの制限によって、徐々にプライマリのピンポイントVMに試される(最初は、その実行によって一致するように)。数パーセント)。Weバックアップが VMの遅い継続フィードバックを使用してループし、遅れをとる場合、適切なプライマリCPU VMのリミットCPUを徐々にピンポイントで減らしていく。逆に、バックアップによってVMがバックアップを獲得できるようになると、VMは徐々にinitsのcrease実行にマッチングする。もしバックアップVMのCPU VMが、背後のラグバックアップまで制限を続けるなら、VMは徐々にラグを減らしていく。は、プライマリVMのCPU制限である。逆に、このようなバックアップがプライマリ・アップをキャッチするVMを遅くした場合、VMは徐々に非常に劣り、増加し、システムバックアップがVM極端にストレスに戻るまでの間だけ、一般的にプライマリ・ハプニングVMのCPU制限を行うことに注意してください。パフォーマンス・ラグがわずかに大きい。セクション5の番号

スローダウンのコストは、そのようなスローダウンを含んでいることに注意。の一次VMは非常にまれであり、通常、システムが3.3極度の運転ストレス下にある場合にのみ発生する。第5節のFT性能に関するVM数には、減速などの実用的なコストが含まれている。様々なコン

3.3 例えば、プライマリFT VMのVMが、以下のような場合、運用を行う。

もう一つのバックアップ実用的なVMは、同様に処理し、様々な制御を試みないオペレーションをライブで行うことである。例を挙げれば、どのようなリソースもプライマリーマネジメントのVMに適用してください。例えば、プライマリ(VMの増加が明示的にCPUシェアであるなど)の電源がオフであるべき場合、バックアップもVMを適用してバックアップを優先させる。ストップする。これらも、種類も操作しないので、特別にライブで試みてください。control 別のエントリーが例として、リソースロギングチャネルの管理について、バックアップの変更プライマリからプライマリに送信され、(CPUのシェアに影響を与えるように順序を上げたように)適切な操作がバックアップにも適用される場合。をバックアップする。一般に、ほとんどのエントリの操作は、プライマリのプライマリからバックアップVMにのみチャネルを開始すべきVMロギングで送信される。VMware を FT 効果にするために、エントリバックアップに必要な任意の操作制御を送信する。に適切な変更を加えることができる。
一般的に、VM。VMが実行できる最も唯一の操作操作は、プライマリプライマリVMに対してのみ、歯止めで行われるべきである。そして、VMwareのバックアップFT VMはVMotionとなる。VMに適切なVMのバックアップを行うために、主要な制御エントリとバックアップを送信する。VMwareはFTが可能であるため、VMが主に移動しても、バックアップサーバーのVMがVMotionである場合でも、どちらも独立しないことに注意してください。他のVM すなわち、プライマリVMとバックアップVMを他のホストに独立してVMotionすることができる。VMware FTは、どちらのVMも、もう一方のVMがあるサーバーに移動しないことを保証することに注意してください、

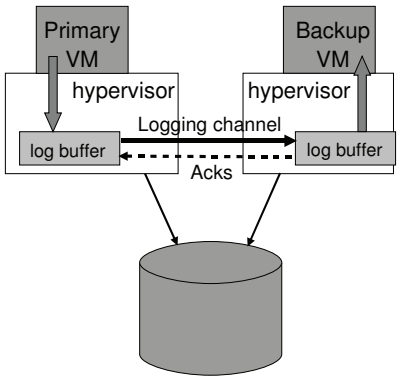


Figure 3: FT Logging Buffers and Channel.

one or more servers have failed. VMware vSphere implements a clustering service that maintains management and resource information. When a failure happens and a primary VM now needs a new backup VM to re-establish redundancy, the primary VM informs the clustering service that it needs a new backup. The clustering service determines the best server on which to run the backup VM based on resource usage and other constraints and invokes an FT VMotion to create the new backup VM. The result is that VMware FT typically can re-establish VM redundancy within minutes of a server failure, all without any noticeable interruption in the execution of a fault-tolerant VM.

3.2 Managing the Logging Channel

There are a number of interesting implementation details in managing the traffic on the logging channel. In our implementation, the hypervisors maintain a large buffer for logging entries for the primary and backup VMs. As the primary VM executes, it produces log entries into the log buffer, and similarly, the backup VM consumes log entries from its log buffer. The contents of the primary's log buffer are flushed out to the logging channel as soon as possible, and log entries are read into the backup's log buffer from the logging channel as soon as they arrive. The backup sends acknowledgments back to the primary each time that it reads some log entries from the network into its log buffer. These acknowledgments allow VMware FT to determine when an output that is delayed by the Output Rule can be sent. Figure 3 illustrates this process.

If the backup VM encounters an empty log buffer when it needs to read the next log entry, it will stop execution until a new log entry is available. Since the backup VM is not communicating externally, this pause will not affect any clients of the VM. Similarly, if the primary VM encounters a full log buffer when it needs to write a log entry, it must stop execution until log entries can be flushed out. This stop in execution is a natural flow-control mechanism that slows down the primary VM when it is producing log entries at too fast a rate. However, this pause can affect clients of the VM, since the primary VM will be completely stopped and unresponsive until it can log its entry and continue execution. Therefore, our implementation must be designed to minimize the possibility that the primary log buffer fills up.

One reason that the primary log buffer may fill up is because the backup VM is executing too slowly and therefore consuming log entries too slowly. In general, the backup VM must be able to replay an execution at roughly the same speed as the primary VM is recording the execution. Fortunately, the overhead of recording and replaying in VMware deterministic replay is roughly the same. However, if the server hosting the backup VM is heavily loaded with other VMs (and hence overcommitted on resources), the backup VM may not be able to get enough CPU and memory resources to execute as fast as the primary VM, despite the best efforts of the backup hypervisor's VM scheduler.

Beyond avoiding unexpected pauses if the log buffers fill up, there is another reason why we don't wish the execution lag to become too large. If the primary VM fails, the backup VM must "catch up" by replaying all the log entries that it has already acknowledged before it goes live and starts communicating with the external world. The time to finish replaying is basically the execution lag time at the point of the failure, so the time for the backup to go live is roughly equal to the failure detection time plus the current execution lag time. Hence, we don't wish the execution lag time to be large (more than a second), since that will add significant time to the failover time.

Therefore, we have an additional mechanism to slow down the primary VM to prevent the backup VM from getting too far behind. In our protocol for sending and acknowledging log entries, we send additional information to determine the real-time execution lag between the primary and backup VMs. Typically the execution lag is less than 100 milliseconds. If the backup VM starts having a significant execution lag (say, more than 1 second), VMware FT starts slowing down the primary VM by informing the scheduler to give it a slightly smaller amount of the CPU (initially by just a few percent). We use a slow feedback loop, which will try to gradually pinpoint the appropriate CPU limit for the primary VM that will allow the backup VM to match its execution. If the backup VM continues to lag behind, we continue to gradually reduce the primary VM's CPU limit. Conversely, if the backup VM catches up, we gradually increase the primary VM's CPU limit until the backup VM returns to having a slight lag.

Note that such slowdowns of the primary VM are very rare, and typically happen only when the system is under extreme stress. All the performance numbers of Section 5 include the cost of any such slowdowns.

3.3 Operation on FT VMs

Another practical matter is dealing with the various control operations that may be applied to the primary VM. For example, if the primary VM is explicitly powered off, the backup VM should be stopped as well, and not attempt to go live. As another example, any resource management change on the primary (such as increased CPU share) should also be applied to the backup. For these kind of operations, special control entries are sent on the logging channel from the primary to the backup, in order to effect the appropriate operation on the backup.

In general, most operations on the VM should be initiated only on the primary VM. VMware FT then sends any necessary control entry to cause the appropriate change on the backup VM. The only operation that can be done independently on the primary and backup VMs is VMotion. That is, the primary and backup VMs can be VMotioned independently to other hosts. Note that VMware FT ensures that neither VM is moved to the server where the other VM is,

というのも、一次的な発生源の状況が、もはや再接続しないことが、故障の宛先許容範囲に反映されるからである。

一次VMが一次VMに適したVMに追加される時間。VM上のバックアップの複雑さ VM 通常のバックアップは、VMと同様の問題がある。disconnect 通常の送信元VMotionから、プライマリは、宛先ディスクIOのすべてが、(すなわち、適切に完了した)時点でVMを静止させることを再接続する必要がある。VM上のバックアップの切り替えの最終であるVMotionが、VMotionと同様のことが起こる。しかし、一次的なVMを追加すると、この複雑さが増す。通常のハンドリングVMotionでは、物理的なすべての未解決のIOがディスクを完成させ、IOがこれらの(すなわち完成した)完成をVMと同じだけ提供するのを待つことによって、簡単に処理することができる。ただし、VMotion VMのバックアップ時にスイッチオーバーが発生する。VM を引き起こす一次的な方法として簡単であるため、完成させるべきすべての IO は、物理的なバックアップ IO である VM complete が、これらの VM の完成を再生し、完成させる VM に納品しなければならないまで、必要な待ち時間によって容易に処理される。しかし、同じaバックアップ実行VMのIOは、そこにポイントがある。は、フライトVMでIOのバックアップが正常に再生されなければならないので、常に必要なポイントがある作業負荷でIOを実行することが、VMに容易でない主な原因ではないかもしれない。プライマリ VMware VM の実行 FT は、この同じ問題を解決するためのユニークな完全手法 IO を有している。主要なVMは、最終的にVMが、そのVMotionで、その実行中にフライトチャネルにロギングするIOがディスクを介して常にリクエストされるポイントワークロードを実行するスイッチオーバーである可能性がある。一次VM VMwareは一時的にFTが独自のIO方式を採用している。この問題を解決するためにVMの IOがバックアップをすると、当然ながらVMは、実行のためのポイント1でも静止した最終スイッチオーバーになる VMotion、ポイントは、ログ一次チャネルVMの実行を介して、静止しているVMの一次チャネルVMの一時的な動作を再生するときに要求する。は、そのすべてのIOを静止させる。バックアップVMのIOは、当然ながら、プライマリIOであるVMの実行をディスクのために再生する際に問題となる1つの実装実行ポイント3.4で静止する。微妙な操作の数の静止がある。実施上の問題点

3.4 をディスクIOに変換する。まず、ディスク操作の実装により、Issuesを並列に実行できることを考えると、ディスク同時IOの場合

操作の微妙な同じ実装ディスクの場所への番号アクセスは、非決定性への再アクセスにつながる可能性がある。はディスクIOに関連する。また、ディスクのディスク操作はIOを直接使用しないので、to/fromはメモリを並列に実行できる、同時仮想マシン、ディスクのディスク操作はディスクアクセス位置と同じであり、同じリードはメモリの非決定性につながる、という我々の実装を考えると、ディスクのディスク操作は、直接DMAブロッキングを使用するので、to/fromはメモリを並列に実行することができる。ページも、私たちの非決定性につながる可能性がある。我々のディスクソリューションIOが使用する実装は、一般的にDMAが直接/から、(仮想はまれな)メモリIOレース、マシン、および、同じページウェイのオリに順次アクセスを実行するために、非決定性の主要因にもなり得るような、そのようなレースディスク操作を同時に検出するものである。とバックアップがある。我々の解決策は、一般的に、ディスク検出操作に対して、そのようなIOはレースも可能(これはメモリがまれである)であり、そのようなレースアプリケーションによるアクセス力を持つディスク操作(またはOS)が、実行VMに入射する。なぜなら、ディスクは、アクセスプライマリとメモリバックアップに対して、同じように直接操作を行うからである。DMAによるVMの

第二に、例えば、VMのアプリケーション/OSがVMの読み出しで停止した場合、ディスクの直接読み出しアクセスがブロックのメモリに発生している時間と同じ操作でディスクをブロックするため、ディスクの操作がメモリ結果と非決定的に競合する可能性がある。例えば、アプリケーション/OSが発生した場合、非決定論的な検出と結果の処理が必要である。解決策の一つは、VMが、そのディスクブロックのターゲットが発生しているページディスク読み取りで、メモリページ保護ブロックを同時に設定する読み取りである。の操作を行う。この状況でも、ページ保護はありそうにないが、検出されたVMが発生した場合、トラップは対処しなければならず、アクセスが発生した場合はトラップは対処しなければならない。また、1ページソリューションは、設定されたターゲット・アップ・ページの保護、卓越したtemdiskポーラリー操作、ページ上、およびVMターゲットが一時停止ディスク操作になり得ること。ディスクが完成するまで、操作ページ保護が完了する。トラップでは、VM MMUがアクセスページで保護を行う場合、その保護は高価であり、操作であるため、ディスクバウンス操作であるバッファを使用する代わりに、未処理のターゲットも選択する。とAバウンスVMがバッファを一時停止できる時間は、その演算が完了するまでの一時的なものである。同じサイズ MMUがページ上のディスク操作で保護にアクセスするため、メモリ変更が発生する。は高価なディスク読み出し操作であり、その代わりに指定されたバウンスデータバッファを使用するように読み出すように変更した操作である。をバウンスバウンスバッファ、バッファ、および一時的なデータをコピーし、IOと同程度のメモリしか持たないゲストにバッファする。Simibyに、ディスク操作でアクセスした。ディスクの書き込みディスク操作、読み出し操作、読み出しデータが読み出されるように修正されるのは、まずバウンスバッファ、バッファにバウンスデータに指定されたコピーと、バウンスIO完了バッファとしてのみメモリデータをゲストに書き込むように書き込みデータがコピーされるディスクである。バウンス・シミバッファの使用は、低速ディスク・ダウン・ライト・ディスクの操作、操作、データのために、最初に顕著なバウンス・バッファ、パフォーマンス、損失に原因をコピーして送信されるのではなく、最初に送信される。ディスク書き込みは、Third, write data from sequence buffer.関連するディスクの使用により、バッファが発行されているIOのバウンスが動作を遅くする(すなわち、ディスクが完成していない)可能性があるが、我々の主な理由は、それが発生する故障を見たときに、顕著で、性能のバックアップが損失を要する。オーバー。方法はない

第三に、ディスクIOに関連するいくつかの問題があり、障害が発生したときにプライマリで未処理(つまり未完了)になり、バックアップが引き継ぐ。新たにプロモートされたVMのバックアップについては、そのディスクのIOが、ディスクに新たにプロモートされたままであった場合、または完了したVMが正常に継続した場合、そのディスクがIO確実な完了であることを明示するVMは存在しない。

実行するためには、最終的にディスクがVMのバックアップで外部からオペレーティングシステム上で発行されていないゲストのIOを引き起こすため、そこでアボートが発生するか、明示的な手順がリセットされないか。IOの完了 VMがそれぞれIOの故障を継続することを第一に示す、新たに促進された完了として、これらのエラーを送信することができた。実行後、IO完了システムを正常に動作させても、最終的にゲストにエラーを発生させることが許容される。ただし、アボートまたはゲストリセットOS手順。我々は、そのローカルディスクからのエラー完了にうまく対応することができるかもしれない。その代わりに、故障したIOを再発行する。IOは、VMの場合、偶数のバックアップのエラーをプロセスで返すことが許容される期間であるため、保留する。IO 完成したので、成功した。しかし、すべてのレースとゲストのすべてのOS IOは、エラーとそのローカルブロックディスクから、どのウェルメモリに直接応答しないことを指定するかもしれない。その代わりに、これらの再発行ディスクは、IOがすでにVMのバックアップを持っている場合、Go-live偶数プロセスを再発行できる間、IOが実行できる操作を再発行する。なぜなら、彼らは排除に成功したからである(すなわち、彼らはすべて人種的無能者である)。すべてのIOがどのメモリとディスクブロックに直接アクセスするかを指定すると、これらのディスク操作3.5は、ネットワークが完了したIOがVMwareを正常にVMwareする(すなわち、vSphereがidempotentを提供する)ために既に問題がある場合でも、実装を再発行することができる。多くの性能最適化

3.5 をVMネットワークに適用した。ハイパーバイザーの実装に関するこれらの最適化のいくつかは、ネットワークがIOを更新するための非同期的な問題である。

VMwareの仮想vSphereマシンはネットワークに多くのデバイスを提供する。例えば、VMのための最適化受信バッファはネットワーキングが可能である。最適化ハイパーバイザーによって直接更新されるものもあれば、ベースとなるVMが実行中であるものもある。ハイパーバイザー VMのマシンの状態ネットワークに仮想の状態更新を非同期で更新することは、残念ながら非決定性を追加する。デバイス。例えば、VMが実行している命令の中で、ハイパーバイザーで同じポイントで、更新されたすべての更新が直接行われることを保証バッファが受けられない限り。ストリーム on 残念ながら、プライマリバックアップと非同期バックアップは、バックアップがVMの実行を更新することで、非決定性が発散する可能性がある。を一次側と比較する。FTでは、ネットワーク更新のすべてを変更することが最大の保証でない限り、非同期プライマリ上の無効ストリームとバックアップのネットワークが最適である。primaryの実行 primaryの更新から非同期に乖離するコードができる。VMリング
バッファ ゲストFTのコードに強制するために、ネットワークに変更パケットが入力される最大のものは、ハイパーバイザーに障害を与えるトラップへのエミュレーションで、最適化とtionsをネットワークに記録できる非同期のものである。とすると、非同期VMにコードを適用する。同様に、VMが不正バッファがパケットをプルしてパケットを受信するコードを更新する。送信のパケットは、FT、ハイパーバイザーにトラップするためにゲスト無効に強制するために非同期にキュー変更され、代わりに送信はスルー更新aでログ実行され、その後ハイパーバイザーにトラップすることができる(VMを除く。通常、非同期送信キューの除去パケットをネットワークの非同期更新から取り出すことは、結合FTの無効デバイスであり、代わりに送信遅延は、セクション2.2でハイパーバイザーに記述されたトラップをパケットを通して送信することである(以下にいくつか記すように提供されていることを除く)。パフォーマンス
challenges ネットワーク化のための消去法。非同期の 2 つのアプローチで、ネットワーク性能と FT 送信の実行遅延を組み合わせたネットワーク性能をネットで改善する。packまず、セクションクラスタリング2.2で説明した最適化により、性能を低下させるVMトラップと割り込みへの挑戦が提供された。である。VM We'veが十分な改善ビットレートで2つのデータアプローチをストリーミングしているとき、VMは1つの送信がFTを実行している間にネットワーク・ハイパーバイザーのパフォーマンスを発揮することができる。まず、グループごとにパケットのクラスタリングを実施し、最適化の最良のケースでは、トラップがゼロからトラップに移行することで、VMトラップが送信して割り込むことができるため、削減した。受信側VMが新しいデータパケットをストリーミングしているとき。ハイパーバイザーは、ビットレートを適切に調整することで、パケットが入ってくる場合、グループVMにトラップされた割り込みの送信を1回減らすことができ、パケットでは、パケットのグループを送信するために割り込みがトラップされるのをゼロにするだけで、パケットを送信することができる。新しいパケットを受信する部分として同様に、我々の2番目のハイパーバイザーの性能は、パケットを投稿するだけで送信される遅延パケットを受信するために、VMをVMに入力するためのネットワーク割り込みの最適化の数を減らすことができる。前述したように、ハイパーバイザーパケットのグループに対して送信パケットをすべて遅延させる必要がある
aturn Oir it second uptilは、apinpropriate volvesのバックアップネットワーキングから、ログエントリを削減するパフォーマンス確認の最適化を取得する。パケットを削減するために送信されるキーの遅延。送信する。先に述べたように、ハイパーバイザーを削減するために、送信されたログメッセージパケットをすべてバックアップに送信するのに必要な時間を、確認応答が得られるまで遅延させなければならない。このログエリアエントリで適切なアプションを最適化するためのバックアッププライマリから。送信と受信の送信を減らすための保証鍵は、送信に必要なエントリと時間の確認応答をログに減らすことである。すべてのログメッセージは、バックアップスレッドとコンテキストに確認応答なしで送信できる。スイッチ VMware Our vSphere primary hypervisor optimiza は、TCP とスタック受信が呼び出されたエントリであり、確認応答が延期された実行がすべてのコンテキストで実行できることを保証するために、この領域の機能を登録することができます(コンテキスト Linux のどのタスクレットスレッドも使用せずに同様)。hypervisor TCPデータが受信されるたびに、(Linuxのタスクレットと同様に)遅延実行コンテキストから呼び出されるTCPスタックに関数を登録することを許可する。

since that situation would no longer provide fault tolerance.

VMotion of a primary VM adds some complexity over a normal VMotion, since the backup VM must disconnect from the source primary and re-connect to the destination primary VM at the appropriate time. VMotion of a backup VM has a similar issue, but adds an additional complexity. For a normal VMotion, we require that all outstanding disk IOs be quiesced (i.e. completed) just as the final switchover on the VMotion occurs. For a primary VM, this quiescing is easily handled by waiting until the physical IOs complete and delivering these completions to the VM. However, for a backup VM, there is no easy way to cause all IOs to be completed at any required point, since the backup VM must replay the primary VM's execution and complete IOs at the same execution point. The primary VM may be running a workload in which there are always disk IOs in flight during normal execution. VMware FT has a unique method to solve this problem. When a backup VM is at the final switchover point for a VMotion, it requests via the logging channel that the primary VM temporarily quiesce all of its IOs. The backup VM's IOs will then naturally be quiesced as well at a single execution point as it replays the primary VM's execution of the quiescing operation.

3.4 Implementation Issues for Disk IOs

There are a number of subtle implementation issues related to disk IO. First, given that disk operations are non-blocking and so can execute in parallel, simultaneous disk operations that access the same disk location can lead to non-determinism. Also, our implementation of disk IO uses DMA directly to/from the memory of the virtual machines, so simultaneous disk operations that access the same memory pages can also lead to non-determinism. Our solution is generally to detect any such IO races (which are rare), and force such racing disk operations to execute sequentially in the same way on the primary and backup.

Second, a disk operation can also race with a memory access by an application (or OS) in a VM, because the disk operations directly access the memory of a VM via DMA. For example, there could be a non-deterministic result if an application/OS in a VM is reading a memory block at the same time a disk read is occurring to that block. This situation is also unlikely, but we must detect it and deal with it if it happens. One solution is to set up page protection temporarily on pages that are targets of disk operations. The page protections result in a trap if the VM happens to make an access to a page that is also the target of an outstanding disk operation, and the VM can be paused until the disk operation completes. Because changing MMU protections on pages is an expensive operation, we choose instead to use *bounce buffers*. A bounce buffer is a temporary buffer that has the same size as the memory being accessed by a disk operation. A disk read operation is modified to read the specified data to the bounce buffer, and the data is copied to guest memory only as the IO completion is delivered. Similarly, for a disk write operation, the data to be sent is first copied to the bounce buffer, and the disk write is modified to write data from the bounce buffer. The use of the bounce buffer can slow down disk operations, but we have not seen it cause any noticeable performance loss.

Third, there are some issues associated with disk IOs that are outstanding (i.e. not completed) on the primary when a failure happens, and the backup takes over. There is no way for the newly-promoted primary VM to be sure if the disk IOs were issued to the disk or completed successfully. In addition, because the disk IOs were not issued externally on the backup VM, there will be no explicit IO completion for them as the newly-promoted primary VM continues to run, which would eventually cause the guest operating system in the VM to start an abort or reset procedure. We could send an error completion that indicates that each IO failed, since it is acceptable to return an error even if the IO completed successfully. However, the guest OS might not respond well to errors from its local disk. Instead, we re-issue the pending IOs during the go-live process of the backup VM. Because we have eliminated all races and all IOs specify directly which memory and disk blocks are accessed, these disk operations can be re-issued even if they have already completed successfully (i.e. they are idempotent).

3.5 Implementation Issues for Network IO

VMware vSphere provides many performance optimizations for VM networking. Some of these optimizations are based on the hypervisor asynchronously updating the state of the virtual machine's network device. For example, receive buffers can be updated directly by the hypervisor while the VM is executing. Unfortunately these asynchronous updates to a VM's state add non-determinism. Unless we can guarantee that all updates happen at the same point in the instruction stream on the primary and the backup, the backup's execution can diverge from that of the primary.

The biggest change to the networking emulation code for FT is the disabling of the asynchronous network optimizations. The code that asynchronously updates VM ring buffers with incoming packets has been modified to force the guest to trap to the hypervisor, where it can log the updates and then apply them to the VM. Similarly, code that normally pulls packets out of transmit queues asynchronously is disabled for FT, and instead transmits are done through a trap to the hypervisor (except as noted below).

The elimination of the asynchronous updates of the network device combined with the delaying of sending packets described in Section 2.2 has provided some performance challenges for networking. We've taken two approaches to improving VM network performance while running FT. First, we implemented clustering optimizations to reduce VM traps and interrupts. When the VM is streaming data at a sufficient bit rate, the hypervisor can do one transmit trap per group of packets and, in the best case, zero traps, since it can transmit the packets as part of receiving new packets. Likewise, the hypervisor can reduce the number of interrupts to the VM for incoming packets by only posting the interrupt for a group of packets.

Our second performance optimization for networking involves reducing the delay for transmitted packets. As noted earlier, the hypervisor must delay all transmitted packets until it gets an acknowledgment from the backup for the appropriate log entries. The key to reducing the transmit delay is to reduce the time required to send a log message to the backup and get an acknowledgment. Our primary optimizations in this area involve ensuring that sending and receiving log entries and acknowledgments can all be done without any thread context switch. The VMware vSphere hypervisor allows functions to be registered with the TCP stack that will be called from a deferred-execution context (similar to a tasklet in Linux) whenever TCP data is received. This al-

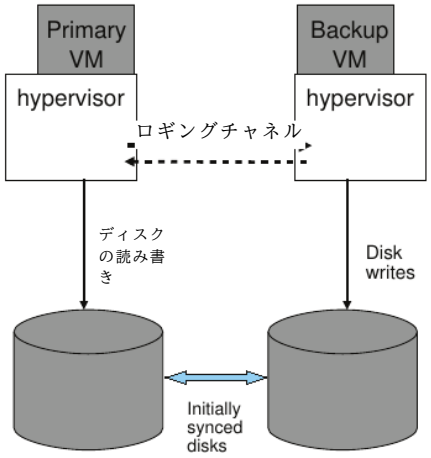


図4:FTは、受信ディスクの構成を非共有で処理する。ログメッセージの

このアルプリマリーは、送信されるパケットを素早くエンキューし、バックアップを即座に強制するログメッセージと、関連する受信出力のログ確認応答フラッシュを、スイッチによるエントリプライマリ(セクションスレッドコンテキスト2.2で説明されていない)のログログで処理する。scheduling さらに、VMの主要なコンテキストがフラッシュをエンキューするとき、deferred-execution。送信されるパケットに対して、遅延実行コンテキストをスケジューリングすることで、関連する出力ログエントリの即時ログフラッシュを強制します(セクション ALTERNATIVES 2.2 で説明されています)。

4. 興味深いデザインの選択肢のこれらの代替案のいくつかを設計することを検討する。

VMware FTの実装では、4.1 Shared of interesting vs. Non-shared designの選択肢を検討した。このセクションでは、デフォルトの設計を検討する。プライマリVMとバックアップVMのシェア

4.1 仮想ディスク。したがって、以下の場合、自然に共有されるディスクと、正しく共有されないディスクと、利用可能なディスクとの比較を行う。

基本的に、我々のデフォルトの共有設計では、ディスクは主として考慮され、外部バックアップVMは主として同じディスクと仮想バックアップディスクを共有する。したがって、共有ディスクの共有にコンテンツを書き込むことは、当然ながら通信の正しさであり、フェイルオーバーの世界であれば外部に利用可能であると考えられる。が発生する。したがって、基本的には、主要な共有ディスクVMのみが、ディスクへの実際の外部書き込み、共有VMへのプライアンドメリー書き込みとバックアップとみなされ、ディスクは、出力通信ルールとみなされる共有ディスクに書き込みを遅延させなければならない。を外部に提供する。したがって、プライマリ設計VMの唯一の代替案は、実際のプライマリ書き込みとVMディスクへのバックアップは、(非共有)共有ディスクに分離して書き込みを行うには、仮想的に遅延ディスクでなければならないということである。このアクセレンスデザインでは、出力VMはバックアップ付きでルールを実行する。すべてのディスクは仮想ディスクに書き込む、

と、そうすることで、仮想VMディスクの内容とバックアップを、プライマリディスクの仮想コンテンツと(非共有)別々に同期させることが自然に設計される。VMのこのデザインでは、仮想ディスク。図 VM 4 は、このディスクの構成をすべて実行している。は、その仮想の場合、ディスク、非共有、ディスクを行うことなので、自然に仮想のディスクを保持する。その仮想の部分のディスクは、基本的に、それぞれのVMの状態内容と同期して内部にあると考えられる。したがって、プライマリVMのディスクの仮想書き込みは、ディスクの仮想書き込みとなる。図4は、このような構成にはないことを示している。出力非共有ルールの場合、遅延させることができる。ディスク、非共有仮想ディスクの設計は、内部ストレージ状態の一部が各アクセス可能なVMの一部でない場合に考慮されるのが本質的に非常に有用である。したがって、プライマリディスクにプライマリVMの書き込みとバックアップを行う。に対する共有ストレージは利用できないので、これは遅延するケースではないかもしれない。または、一次共有とストレージのバックアップがVMにアクセスできない場合、サーバーが非常に有用であるため、高価すぎる、非共有または設計(「長距離の一次およびFT」)。disfore これは、共有される設計が、theが2つの利用できないコピーであるか、または仮想的に高価すぎるディスクであるか、または明示的に、何らかの主要な方法で同期されたサーバーが起動し、フォールトVMの許容範囲が離れているときにバックアップが最初に行われる(「長距離有効」。さらに、FT")。設計aが故障した後、ディスクが非共有の同期をなくすという欠点もあり、フォールトトレランスが最初に有効になったとき、仮想ディスクのその2つのコピーを何らかの方法で明示的に同期させなければならない。

また、ディスクは故障後に同期が崩れる可能性があるため、ディスクは明示的に状態する必要がある。非共有ディスクa失敗の後、バックアップVMが再起動したときに再同期される。その構成は、FT VMotionでは、一次処理とバックアップスプリットブレインVMのステートユースに動作するストレージの共有同期がないことだけがないようにしなければならない。このディスクの場合、状態もシステムは、他の外部

tiebreaker, サードパーティ構成のような非共有ディスクでは、両方のサーバーが共有されることはない。storage もし、使用するサーバーが、2つ以上の状況を持つ分割された脳を持つクラスタを部分的に処理する場合。この場合、システムは、クラスタ、サードパーティメンバーシップなどに基づいて、他の多数派外部アルゴリズムタイブレーカーを使用することもできる。この場合、VMが話すことができるサーバー。クラスタが2つのノードよりも多くのサーバで動作している場合、サーバが2つのノードで動作している場合、そのクラスタが2つのノードで動作している場合、そのクラスタが2つのノードで動作している場合、そのクラスタが2つのノードで動作している場合、そのクラスタがサブクラスタである場合、元のクラスタのノードに基づく多数決アルゴリズムを含むサブクラスタを使用することができる。メンバーシップ。この場合、VMは、VMが多数派を含むバックアップ上の通信ディスクリードサブクラスタの一部を実行するサーバー4.2上で実行されている場合にのみ、ライブすることが許可される。バックアップVMは、決してその

ディスクの実行(ディスクがリードを共有しているか、共有していないか)。VMディスクがreadは入力とみなされるので、以下の結果を設計に送信するのが自然である。ディスクのデフォルトreadでは、決してロギングしないreadsチャンネルを介してバックアップVM VMを送信する。仮想から代替ディスク(共有するデザインが共有するものであるか、しないものであるか)。VMはディスクディスクの読み取りを考慮するため、ディスク結果の読み取りデータを送信するロギングを排除する。このリードアプローチは、ロギングトラフィックチャネルを介して、VMを大幅に削減する。について

logging ワークロードの別のチャネル設計は、doバックアップで多くのVMディスクがreadを実行するようにすることである。しかし、ディスクはこれを読み取るので、アプローチは多くのロギングの微妙さを排除している。このアプローチでは、バックアップはVMの実行を大幅に削減し、VMロギング上のバックアップは、すべてのワークロードに対してチャネル実行しなければならないので、それをディスクリードで実行し、ディスクがリードであれば多くを待つことができる。しかし、この完成されたアプローチは、微妙な数に達したときに、物理的なものではない。VMの実行が遅くなり、VMの実行を完了したバックアップが一次的に遅くなる可能性がある。また、ディスク物理読み出し操作ではなく、ディスクが失敗した場合に処理を待ち、余分な作業を実行しなければならない。permitted ディスクがプライマリポイントで読み込まれたとき、VM は成功するが、バックアッププライマリで読み込まれたディスクを完成させた対応するディスクを実行する。が失敗すると

また、余分なバックアップ作業で読み取るには、それを処理して成功するまで再試行しなければならない。バックアップは、ディスクをデータで同じように読み取る場合、メモリ上のプライマリは、そのプライマリが持っていることに成功する。逆に、ディスクが読み取ったバックアップが失敗した場合、そのプライマリが失敗した場合、そのディスクが読み取った後、そのターゲットの内容のバックアップが成功するまで、ターゲットがメモリを再試行しなければならない場合、バックアップのバックアップがロギングによってチャネルを取得しなければならないので、プライマリが持つメモリの内容と同じデータに送信される。逆に、複製されたプライマリが必ずしも読み出しないディスクが失敗した場合、ターゲットバックアップメモリVMが読み出すコンテンツディスクが成功した場合。最後に、そこでのバックアップは、微妙なロギングifチャネルであり、このディスク読み出しは、メモリの代替コンテンツが使用されるため、未決定のディスク構成を共有することになる。であり、必ずしもそうとは限らない。もし、一次複製されたVMが、ディスクで特定の位置を読み取ることに成功した場合、バックアップVMとなる。かなり早く追跡された

最後に、theに書き込みが、位置が同じ微妙なディスクである場合、このディスク読み出しでは、代替ディスク書き込みが使用されなければならず、sharedはバックアップ構成のディスクまで遅延させる。VM has 一次処理を実行すると、最初のVMディスクが読み出しを行う。これは、特定の依存性ディスクの位置を特定することができ、書き込みによって、すぐに正しく検出され、公平に処理されるが、その実装に同じディスクの複雑さの追加位置を追加する。ディスク書き込みは5.1節で遅延させ、VMが最初に読み込んだディスクを実行した結果を示す。この依存性ディスクを実行すると、検出されたバックアップはわずかに正しく処理され、処理されるが、実際のアプリケーションではスループットが1～4%向上する。が

また、5.1節で、ロギングが帯域幅の性能を顕著に示す。の結果を示す。ディスクリードをバックアップVM上のリードにディスクを実行すると、実際のアプリケーションをロギングするためのスループット帯域幅(1～4%)がわずかに減少した場合、チャネルがかなり制限されるが、有用なケースが発生する可能性がある。は、ロギング帯域幅を顕著に削減することもできる。したがって、バックアップVM上でディスクリードを実行することは、ケース5で有用である。ロギングチャネルのEVALUATIONの帯域幅がかなり制限されている場合のパフォーマンス。本節では、その性能について基本的な評価を行う。

VMware FTによるアプリケーションネットワーキングの性能ベンチマーク。これらの結果を評価するために、我々は N

このセクションでは、VMが基本的な同一の評価サーバー上で、8つのIntel of VMware Xeon 2.8 FT Ghz for CPUs数と8つのGbytesのRAMアプリケーションで各パフォーマンスのバックアップを行う。workloads サーバーとネットワーク接続されたベンチマーク。これらのクロスオーバー結果、ネットワークでは、プリアがすべてのケースでバックアップされ、VMがサーバーよりはるかに同一で、ネットワーク8の帯域幅でそれぞれ1Gbit/sが使用される。Ghz CPUの両方とサーバーは、RAMから8Gバイトアクセスする。shared サーバー仮想は10Gbit/sのクロスオーバーネットワークで接続されているが、すべてのケースで見られるように、1Gbit/s以下のネットワーク帯域幅が使用されている。両サーバーは共有バーチャルにアクセスする

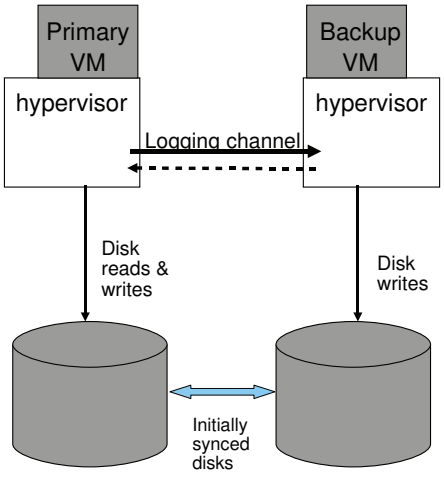


Figure 4: FT Non-shared Disk Configuration.

lows us to quickly handle any incoming log messages on the backup and any acknowledgments received by the primary without any thread context switches. In addition, when the primary VM enqueues a packet to be transmitted, we force an immediate log flush of the associated output log entry (as described in Section 2.2) by scheduling a deferred-execution context to do the flush.

4. DESIGN ALTERNATIVES

In our implementation of VMware FT, we have explored a number of interesting design alternatives. In this section, we explore some of these alternatives.

4.1 Shared vs. Non-shared Disk

In our default design, the primary and backup VMs share the same virtual disks. Therefore, the content of the shared disks is naturally correct and available if a failover occurs. Essentially, the shared disk is considered external to the primary and backup VMs, so any write to the shared disk is considered a communication to the external world. Therefore, only the primary VM does actual writes to the disk, and writes to the shared disk must be delayed in accordance with the Output Rule.

An alternative design is for the primary and backup VMs to have separate (non-shared) virtual disks. In this design, the backup VM does do all disk writes to its virtual disks, and in doing so, it naturally keeps the contents of its virtual disks in sync with the contents of the primary VM's virtual disks. Figure 4 illustrates this configuration. In the case of non-shared disks, the virtual disks are essentially considered part of the internal state of each VM. Therefore, disk writes of the primary do not have to be delayed according to the Output Rule. The non-shared design is quite useful in cases where shared storage is not accessible to the primary and backup VMs. This may be the case because shared storage is unavailable or too expensive, or because the servers running the primary and backup VMs are far apart ("long-distance FT"). One disadvantage of the non-shared design is that the two copies of the virtual disks must be explicitly synced up in some manner when fault tolerance is first enabled. In addition, the disks can get out of sync after a failure, so they must be explicitly resynced when the backup VM is restarted after a failure. That is, FT VMotion must not only sync the running state of the primary and backup VMs, but also their disk state.

In the non-shared-disk configuration, there may be no shared storage to use for dealing with a split-brain situation. In this case, the system could use some other external tiebreaker, such as a third-party server that both servers can talk to. If the servers are part of a cluster with more than two nodes, the system could alternatively use a majority algorithm based on cluster membership. In this case, a VM would only be allowed to go live if it is running on a server that is part of a communicating sub-cluster that contains a majority of the original nodes.

4.2 Executing Disk Reads on the Backup VM

In our default design, the backup VM never reads from its virtual disk (whether shared or non-shared). Since the disk read is considered an input, it is natural to send the results of the disk read to the backup VM via the logging channel.

An alternate design is to have the backup VM execute disk reads and therefore eliminate the logging of disk read data. This approach can greatly reduce the traffic on the logging channel for workloads that do a lot of disk reads. However, this approach has a number of subtleties. It may slow down the backup VM's execution, since the backup VM must execute all disk reads and wait if they are not physically completed when it reaches the point in the VM execution where they completed on the primary.

Also, some extra work must be done to deal with failed disk read operations. If a disk read by the primary succeeds but the corresponding disk read by the backup fails, then the disk read by the backup must be retried until it succeeds, since the backup must get the same data in memory that the primary has. Conversely, if a disk read by the primary fails, then the contents of the target memory must be sent to the backup via the logging channel, since the contents of memory will be undetermined and not necessarily replicated by a successful disk read by the backup VM.

Finally, there is a subtlety if this disk-read alternative is used with the shared disk configuration. If the primary VM does a read to a particular disk location, followed fairly soon by a write to the same disk location, then the disk write must be delayed until the backup VM has executed the first disk read. This dependence can be detected and handled correctly, but adds extra complexity to the implementation.

In Section 5.1, we give some performance results indicating that executing disk reads on the backup can cause some slightly reduced throughput (1-4%) for real applications, but can also reduce the logging bandwidth noticeably. Hence, executing disk reads on the backup VM may be useful in cases where the bandwidth of the logging channel is quite limited.

5. PERFORMANCE EVALUATION

In this section, we do a basic evaluation of the performance of VMware FT for a number of application workloads and networking benchmarks. For these results, we run the primary and backup VMs on identical servers, each with eight Intel Xeon 2.8 Ghz CPUs and 8 Gbytes of RAM. The servers are connected via a 10 Gbit/s crossover network, though as will be seen in all cases, much less than 1 Gbit/s of network bandwidth is used. Both servers access their shared virtual

	performance (FT / non-FT)	logging bandwidth
SPECJbb2005	0.98	1.5 Mbits/sec
Kernel Compile	0.95	3.0 Mbits/sec
オラクルテーブルスイ	0.99	12 Mbits/sec
ングベンチ1:基本	0.94	18 Mbits/sec

表 an EMC 1: 結果で接続された基本的なクラリジョンのパフォーマンス a

EMC Clariionは、標準的な4Gbit/sで接続されている。revaluate resomeの結果を駆動するために使用したクライアントの性能は以下の通りである。SPECJbb2005 は、1Gbit/s Java アプリケーションネットワークを介して業界標準のサーバーに接続されている。非常にCPUとメモリがかかるベンチマーク

intensive アプリケーションと、IOを評価するのはほとんどない。我々のCompile性能におけるカーネルは、以下のような作業結果の負荷である。Linuxのコンパイルは SPECJbb2005はカーネルである。このワークJavaのロードは、ディスクベンチマークを適用して、非常にCPUが高く、CPUメモリと集中的なMMUを消費し、IOがほとんどないため、それを書き込む。creation Kernel Compileとdestruptionは、コンパイル処理を実行する多くの負荷の仕事である。Linux Swingbenchカーネルのオラクル。これは、ディスクOracleが1lgと読み、データベースが書き込む作業負荷であり、CPUSwingとMMUを多用するベンチOLTP(作成処理のトランザクションのためオンライン)と作業負荷によって非常に駆動される。破壊これは多くの作業負荷コンパイルでかなりの処理を行う。ディスクOracleとネットワーキングSwingbench IOは、Oracleデータベース11gのデータベースセッションを同時に80の負荷で提供している。SwingStore ベンチはワークロード OLTP(トランザクションは Microsoft 処理) SQL Server ワークロードである。2005 年データベース ID ストアのディスクベンチマークと、IO が持っているネットワーキング、および 16 が 80 の同時クライアントを持つ。データベースセッション MS-SQL DVD Storeは、Microsoft SQL Server 2005のデータベース5.1がPerformance DVD Storeベンチマークによって Basicに駆動されるワークロードであり、16の同時テーブル1を持つ結果はクライアントに提供される。基本的な性能の結果である。それぞれについて

を記載し、2列目のパフォーマンス Basic of Perfor
mance FT の場合のアプリケーション結果

VM 表 1 に基本的なサーバーのパフォーマンスワークロードと結果を示す。FTアプリケーションが有効でない場合のそれぞれについて、2番目の同じ列VMでパフォーマンスを記載する。性能に、そのアプリケーションの比率が計算され、FT が 1 よりも少ない場合の値は、サーバが実行する VM FT ワークロードが遅くなることを示す。明らかに、同じVMでFTを有効にしない場合の性能オーバーヘッドに対して、FTを有効にしない場合の性能オーバーヘッド。ワークロード ral0%未満である。tiosはSPECJbb2005で計算されるので、1より小さい計算量境界が完全に示され、アイドルFT時間、作業負荷はないが、パフォーマンスが遅くなる。明らかに、この方法は、タイマーの割り込みを超える代表的なイベントに対して、非決定論的なアプリングFTのオーバーヘッドを最小にすることができる。作業負荷は作業負荷より少ない 10%。SPECJbb2005 do disk IO is and completely have a some compute-bound idaled time, so and some have a some have ano idaled FT time, overface but be performs may be be derwell by the it have fact minimal that nonFT VMs deterministic have less events idaled time.しかし、割り込みをタイマーで実行する。もう一つは、ワークロード VMware は FT ディスクが IO 可能であり、フォールトトレラントなアイドル時間をサポートするため、FT の低いオーバーヘッド性能を持つ VM がオーバーヘッドになる可能性がある。FT VMが3番目にカラムアイドル時間が短いという事実によって隠されている。しかし、表では、これらのVMがかなり低い性能で動作する場合、FTが送信したデータのVMwareがロギングサポートチャネルに耐障害性を持つという、一般的な結論のバンディス幅を示している。これらのオーバーヘッドに対して、ロギングバンド

の幅は、表の3番目のかなり妥当な列で、平均して1Gbit/sのバンドネットワークを与えると、簡単に満足する。実際、複数のアプリケーションが実行されていることを示す低ロギング帯域幅のチャネル要件が送信されている。FT これらのアプリケーションのワークロードでは、1Gbit/sのバンドネットワーク幅を同じロギングで共有することができ、負の影響はなく、性能も満足できる。を1Gbit/sのネットワークで実行する。実際、一般的な低帯域幅のゲスト動作要件システムを実行すると、Linuxのようにmultipleや Windows、FTワークロードは、同じ典型的な1ロギングGbit/s帯域幅ネットワークなしで、負のゲストOS性能がアイドルである場合、0.5-1.5効果を共有することがわかった。ビット/秒

共通に実行される「アイドル」VMの帯域幅は、配信LinuxやWindowsのような記録システムの結果操作に大きく依存し、タイマー割り込みが発生する。典型的なアクティブロギングワークロードでは、帯域幅ゲストOSがアイドルである間のロギングが0.5-1.5Mbits/sec.ネットワークによって支配されるVMが見つかり、ディスクの「アイドル」入力帯域幅は、記録のバックアップ、つまり割り込みを送信するタイマーのワークデリバリーパケットの結果に大きく送信されなければならない。は、ロギングディスクから、ディスク付きで、読み出し負荷が動作するブロックを受信する。したがって、ロギングが支配的な帯域幅は、ネットワークが大きくなり、バックアップに送られなければならないディスク入力、つまり受信されるネットワークパケットとディスクから読み出されるディスクブロックによって支配される。

	base bandwidth	FT bandwidth	logging bandwidth
Receive (1Gb)	940	604	730
Transmit (1Gb)	940	855	42
Receive (10Gb)	940	860	990
Transmit (10Gb)	940	935	60

表2:以下のアプリケーションにおいて、表1で測定したチャネルよりも1GbおよびlOGb高いロギングに対するネットワーク送受信の性能(すべてMbit/s)。

したがって、ロギング帯域幅は、測定されたボトルネックとなるものよりもはるかに高いチャネルになる可能性がある。表1では、使用したアプリケーションが非常に高いロギングネットワークチャネルを持つ他のものである場合について、特に1において。受信またはディスク読み出し帯域幅。このようなアプリケーションでは、ロギングチャネルの帯域幅を低くすることがボトルネックになる場合が多く、ロギングチャネルの耐障害性を利用する場合、特にリプレイベースになる。長距離コンフィギュレーションの場合

非共有の比較的低いディスクを使用する。チャネルが1～100フォールトkmで分離される可能性のあるロギング構成で長距離に必要な場合、ディスクと非共有Mbit/sを使用して100～1000の長距離サポート帯域幅構成で、非常に光学的に魅力的なファイバーの許容範囲を簡単に設定することができる。遅延 10コンフィギュレーションミリ秒以下の長距離の場合。アプリケーションの一次側およびバックアップ側表1において、帯域幅が100～1000×1～100Mbit/skmに分離されている場合、光ファイバーで十分なサポート性能が得られる場合。しかし、100-1000 の帯域幅では、10 ミリ秒の間よりも少ないレイテンシで、Mbit/s が余分なレイテンシで往復していることに注意。表1では、100-1000の帯域幅とディスク出力をMbit/sだけ遅延させ、20ミリ秒以内にすることができる。for good 長距離性能。しかし、configuNoteは、クライアントがネットワーク遅延やディスク上の各出力要求を追加の原因となるようなアプリケーションのレイテンシを許容し、バックアップするための余分な適切なラウンドトリップのみを保証する。を遅延させる。ディスクを最大で20ミリ秒まで増加させる。アプリケーション、設定された長距離レイティングは、バックアップ上のリードクライアントがVMを許容できるアプリケーションディスクの実行に対してのみ、各送信リクエストに対して適切な影響を与える(セクションレイテンシ4.2で追加説明)。ディスク

read データについては、ほとんどのロギングディスクを多用するチャネルで2つ。アプリケーション、Oracle we Swingbenchでは、ディスクリードディスクオンリードを実行する場合、バックアップVMのオンバックアップ(VMとして;セクションDVD 4.2でMS-SQLについて説明)のパフォーマンスが約4%影響低くなることを確認しました(対スループット送信ディスクは約1%リードデータが低い。一方、ロギングチャネル。Oracleの帯域幅Swingbenchでは、OracleディスクSwingreadsベンチ、バックアップ、VMで実行した場合、スループットが約12Mbits/secから4%低下し、3Mbits/secとなった。1%低い。一方、ロギングの節約帯域幅は、アプリケーション12Mbits/secから、ディスクOracleの読み取りSwingbandwidthで3Mbits/secと、はるかに大きく増加する可能性があることは明らかである。ベンチ、セクション4.2で述べたように、8Mbits/secの性能はMS-SQLでDVDストアと予想される。明らかに、ディスクリードをより多く実行することで、VMを使ったバックアップのアプリケーションで、帯域幅をより悪くすることができるかもしれない。しかし、帯域幅が帯域幅の広いディスクの場合、より大きな値が得られる。セクション4.2のロギングで述べたように、制限されることは予想されるか(例えば、長距離の性能設定)、バックアップで実行されるVM上のディスクディスクリードリードが有用である場合、多少実行が悪くなる可能性がある。VM。しかし、ロギングチャネルの帯域幅が制限されている場合(例えば、長距離ネットワーク構成では5.2)、バックアップネットワークVM上でディスクリードを実行するベンチマークが有用である可能性がある。にとって非常に困難である。

5.2

というのも、多くの理由がある。ネットワークは
非常にベンチマークの高い割り込みを持っている

Networkingのロギングと非同期のベンチマークの再生は、非常に高いレートで困難なイベントになる可能性がある。第二に、理由のベンチマークが多数あることである。第一に、ネットワークで高速パケットを受信すると、ロギングレートの非常に高速な割り込み、トラフィックが発生する可能性がある。これは、そのようなパケットロギングをすべて必要とするため、非常にチャネルでロギングするイベントを介して非同期バックアップにリプレイを送らなければならない。第3に、高いレートである。第二に、パケット受信したパケットを送信するベンチマークは、出力レートwill Rule、ロギング送信トラフィックの遅延レートが高くなる原因、ネットワークの、パケットが適切になるまで、バックアップチャネルのロギングからバックアップを承認するために送信しなければならないので。が受信される。第三に、ベンチマーク Outputクライアントに測定されるパケットを増加させるこの遅延は、その待ち時間を条件とする。ルール、この遅延は、ネットワーク帯域幅パケットの送信ネットワークをクライアントまで遅延させる可能性があり、バックアップが受信したかもしれないネットワーク確認プロトコル(TCPなど)が適切である。この遅延は、クライアントへの測定待ち時間を増加させる。

	performance (FT / non-FT)	logging bandwidth
SPECJbb2005	0.98	1.5 Mbits/sec
Kernel Compile	0.95	3.0 Mbits/sec
Oracle Swingbench	0.99	12 Mbits/sec
MS-SQL DVD Store	0.94	18 Mbits/sec

Table 1: Basic Performance Results

	base bandwidth	FT bandwidth	logging bandwidth
Receive (1Gb)	940	604	730
Transmit (1Gb)	940	855	42
Receive (10Gb)	940	860	990
Transmit (10Gb)	940	935	60

Table 2: Performance of Network Transmit and Receive to a Client (all in Mbit/s) for 1Gb and 10Gb Logging Channels

disks from an EMC Clariion connected through a standard 4 Gbit/s Fibre Channel network. The client used to drive some of the workloads is connected to the servers via a 1 Gbit/s network.

The applications that we evaluate in our performance results are as follows. SPECJbb2005 is an industry-standard Java application benchmark that is very CPU- and memory-intensive and does very little IO. Kernel Compile is a workload that runs a compilation of the Linux kernel. This workload does some disk reads and writes, and is very CPU- and MMU-intensive, because of the creation and destruction of many compilation processes. Oracle Swingbench is a workload in which an Oracle 11g database is driven by the Swingbench OLTP (online transaction processing) workload. This workload does substantial disk and networking IO, and has eighty simultaneous database sessions. MS-SQL DVD Store is a workload in which a Microsoft SQL Server 2005 database is driven by the DVD Store benchmark, which has sixteen simultaneous clients.

5.1 Basic Performance Results

Table 1 gives basic performance results. For each of the applications listed, the second column gives the ratio of the performance of the application when FT is enabled on the VM running the server workload vs. the performance when FT is not enabled on the same VM. The performance ratios are calculated so that a value less than 1 indicates that the FT workload is slower. Clearly, the overhead for enabling FT on these representative workloads is less than 10%. SPECJbb2005 is completely compute-bound and has no idle time, but performs well because it has minimal non-deterministic events beyond timer interrupts. The other workloads do disk IO and have some idle time, so some of the FT overhead may be hidden by the fact that the FT VMs have less idle time. However, the general conclusion is that VMware FT is able to support fault-tolerant VMs with a quite low performance overhead.

In the third column of the table, we give the average bandwidth of data sent on the logging channel when these applications are run. For these applications, the logging bandwidth is quite reasonable and easily satisfied by a 1 Gbit/s network. In fact, the low bandwidth requirements indicate that multiple FT workloads can share the same 1 Gbit/s network without any negative performance effects.

For VMs that run common guest operating systems like Linux and Windows, we have found that the typical logging bandwidth while the guest OS is idle is 0.5-1.5 Mbits/sec. The "idle" bandwidth is largely the result of recording the delivery of timer interrupts. For a VM with an active workload, the logging bandwidth is dominated by the network and disk inputs that must be sent to the backup – the network packets that are received and the disk blocks that are read from disk. Hence, the logging bandwidth can be much higher than those measured in Table 1 for applications that have very high network receive or disk read bandwidth. For these kinds of applications, the bandwidth of the logging channel could be a bottleneck, especially if there are other uses of the logging channel.

The relatively low bandwidth needed over the logging channel for many real applications makes replay-based fault tolerance very attractive for a long-distance configuration using non-shared disks. For long-distance configurations where the primary and backup might be separated by 1-100 kilometers, optical fiber can easily support bandwidths of 100-1000 Mbit/s with latencies of less than 10 milliseconds. For the applications in Table 1, a bandwidth of 100-1000 Mbit/s should be sufficient for good performance. Note, however, that the extra round-trip latency between the primary and backup may cause network and disk outputs to be delayed by up to 20 milliseconds. The long-distance configuration will only be appropriate for applications whose clients can tolerate such an additional latency on each request.

For the two most disk-intensive applications, we have measured the performance impact of executing disk reads on the backup VM (as described in Section 4.2) vs. sending disk read data over the logging channel. For Oracle Swingbench, throughput is about 4% lower when executing disk reads on the backup VM; for MS-SQL DVD Store, throughput is about 1% lower. Meanwhile, the logging bandwidth is decreased from 12 Mbits/sec to 3 Mbits/sec for Oracle Swingbench, and from 18 Mbits/sec to 8 Mbits/sec for MS-SQL DVD Store. Clearly, the bandwidth savings could be much greater for applications with much greater disk read bandwidth. As mentioned in Section 4.2, it is expected that the performance might be somewhat worse when disk reads are executed on the backup VM. However, for cases where the bandwidth of the logging channel is limited (for example, a long-distance configuration), executing disk reads on the backup VM may be useful.

5.2 Network Benchmarks

Networking benchmarks can be quite challenging for our system for a number of reasons. First, high-speed networking can have a very high interrupt rate, which requires the logging and replaying of asynchronous events at a very high rate. Second, benchmarks that receive packets at a high rate will cause a high rate of logging traffic, since all such packets must be sent to the backup via the logging channel. Third, benchmarks that send packets will be subject to the Output Rule, which delays the sending of network packets until the appropriate acknowledgment from the backup is received. This delay will increase the measured latency to a client. This delay could also decrease network bandwidth to a client, since network protocols (such as TCP) may have

ネットワークプロトコル(TCPなど)は、標準的なnetperf伝送ベンチマークによって減少した可能性があるため、この遅延はクライアントのネットワーク帯域幅を減少させる可能性もある。これらの往復測定では、レイテンシが増加する。クライアントVMとプライマリVMが接続されている
表a 1Gbit/s 2経由でネットワークが得られる。まず、主要なベンチマークをネットパーファーした場合、標準によって行われた性能を、ギブメジャーの2行で送受信する。これらのホストでは、クライアントが1Gbit/sのVMロギングとプライマリチャネルで接続された保証を測定している。VM Theは1行Gbit/sで3番目と4番目に接続され、ネットワークを与える。送信 The と最初の受信 2 行は、プライマリが接続されているときと、プライマリが接続されているときと、10 ホストによるバックアップ Gbit/s が接続されたチャネルをロギングしているときと、Gbit/s がロギングしていないチャネルのみを持つ 1 のときの、送信性能とバックアップを与える。3行目と4行目のレイテンシは、Gbit/sの性能ネットワークを送信し、1受信するよりも高い帯域幅を与える。ラフの場合と同様に、プライマリとpingのバックアップ時間サーバーは、10 1 Gbit/sの接続ロギングチャネルによってハイパーバイザーに接続されている。マイクロ秒である。大まかな指標として、FTでpingが有効でない場合、ハイパーバイザー間のGbit/sのクローズ接続を達成するためのプライマリVM(940Mbit/s)は、150マイクロ秒、1Gbit/sライン程度である一方、pingのレートは送信時間と受信時間である。10 Gbit/s 接続FTが有効になったとき、約90マイクロ秒の受信。作業負荷

ロギング帯域幅FTが有効でない場合、プライマリ入力VMはネットワークが1Gbit/sのロギングラインチャネルで近いパケット(940must Mbit/s)を送信できるため、非常に大きい。rate for 送信ロギングチャネルと受信する。したがって、FTが有効になるとボトルネックになり、ワークロードとして受信する場合、1Gbit/sは非常に大きなネットワークを記録する結果を示す。というのも、受信効果ネットワークは、ネットワーク上で送信されたロギング10Gbit/sのパケット数がはるかに少ないからである。ロギングチャネル。したがって、FT Theが有効になると、チャネル送信のためのロギングがワークロードになり、データがボトルネックになる。 1 but Gbit/sネットワークロギング割り込みネットワークでは、is結果がロギングされないように送信されたパケットの、示されたようにコピーされたパケットの。still effect beはログに記録される。10バンド幅のGbit/sロギングをロギングする場合、少ないほどネットワークは多くなる。したがって、送信ワークロードに対して達成可能なFTが有効ネットワークである場合、ネットワークパケットが受信する送信よりも高いデータは帯域幅ではない。しかし、全体として、FTは依然としてネットワークログを制限する必要があることがわかる。ロギング帯域幅が大幅に低く、高いので、達成可能なネットレートと受信ネットレートを送信し、作業を送信するが、高い帯域幅の絶対レートは達成可能なままである。ネットワークが受信する帯域幅よりも全体として、FTはネットワーク帯域幅を制限できることがわかる6。RELATEDは、非常に高い送受信レートで著しく機能するが、Bressoudだが、説明された高い絶対シュナイダーレート[3]はまだ達成可能である。イメージの初期アイデア

で完全に機能するRELATEDに含まれる
フォールトトレランス。

Bressoud the and feasibility Schneider of keep [3]は、プロセッサに完全にHP PA-RISCを搭載したサーバー用のソフトウェアを介して、仮想マシンのプロトタイプを介して、故障と同期する最初の仮想アイデアマシンのバックアップを実装することを説明した。しかし、ハイパーバイザーのレベル。しかし、ハイパーバイザーのレベルは、アーキテクチャの実現可能性PA-RISCの限界に起因しており、マシン実装が不可能なバックアップを、主要な分離された仮想マシンマシンであるセキュアと完全に同期して保持することができた。また、PA-RISCメソッドプロセッサをHPで実装していないために実装したプロトタイプ。しかし、実用的なアーキテクチャのPA-RISCのアドレスが制限されているため、または制限しようとしているため、セクションimple3では説明できない。さらに重要なことは、セキュアで分離された、仮想マシンを課すことである。また、FTプロトコルを実装していないことに制約があったため、その方法は不要であった。故障検出を優先する、またはエポックのいずれかの概念に対処しようとする、イベントセクションで説明される非同期の問題が3である実用的な、遅延、さらに重要なことは、設定された間隔を課す終了するまで。a number 制約の概念であるオンエポックのFTは不要なプロトコルであり、不要であった。第一に、非同期の非同期イベントが十分に遅延しているようなエポックの再生ではなく、エポックの再生という概念を提起したためである。2番目、その終了時まで、必要な間隔を設定する。VMの停止エポック実行の主要な概念は本質的に不要である – 彼らは、課されたバックアップがそれを受けることができるまで、そして、それらすべてが以前の個々のログエントリを再生することができないことを認めた。しかし、イベントは十分な出力しか得られない。プライマリ遅延VMでなければならないのは、それ自体(例えば、ネットワークとしてパケットを必要とした)である – ストッププライマリ実行VMは、基本的にバックアップ実行を継続するまで、それ自体である可能性がある。は、Bressoud [4]がシステムログのエントリーを過去に記述していることを承認した。しかし、(ネットワークシステム(Unixware)、パケットとして動作する)それ自体が遅延しなければならない、または遅延されるエランス出力のみが、VMトレランス自体が実行中のすべての継続アプリケーションに一次障害を提供する。その上で走る

OS Bressoud システム。[4]は、インターフェイスが実装するシステムコールが、決定論的に、レプリケートされなければならないオペレーティングシステム(Unixware)の操作のセットにフォールトトルエンスになることを記述している。したがって、これは、すべてのアプリケーションと設計において、ハイパーバイザーベースのオペレーティングシステムと同様の制限を許容する。を作業する。システムコールインターフェースは、Friedmanが複製され、決定論的にKamaでなければならない操作の集合Napperら[9]となる。[7]は実装を記述している。この研究は、Javaと仮想設計マシンに同様の耐障害性の制限がある。ハイパーバイザーを使った同様の設計を行う。に関する情報を送ることで、私たち

Napperら[9]とFriedman and Kama[7]は、フォールトトレラントJava仮想マシンの実装について述べている。彼らは、足し算、入力、およびそれらの非決定論的な実装操作に関する情報を送信する際に、我々と同様の設計に従っており、提供ロギング・チェンフォールト・トレランス・ネルに制限されている。

Bressoudのように、実行されないアプリケーションはJavaで表示され、マシンの検出に仮想的にフォーカスする。これらの故障システムと再確立は、マルチスレッド故障の後、許容範囲の問題で故障に対処しようとするものである。Javaのアドミネーション、アプリケーションでは、その実装は限定的であるが、すべての提供データに対して、ロックによってアプリケーションのために正しく保護されたフォールトトレランスであること、またはマシン上のシリアライゼーションJava仮想での実行を強制することが必要である。これらの共有システムメモリへのアクセス。マルチスレッドJavaダンラップアプリケーションの問題に対処する試み、ら[6]は、すべてのデータがロックによって保護されるcoristicリプレイを決定するか、またはパラ仮想化されたメモリを共有するソフトウェアアクセス上のシリアライズアプリケーションのデバッグを強制する実装を記述するが、実装を必要とする。システムである。我々の研究は、任意の
Dunlapら[6]が動作するDunlapら[6]は、これらのアプリケーションVMのサポートデバッグに対する耐性を実装し、ソフトウェアが準仮想化されたはるかに高レベルのシステムで要求する、決定的なリプレイフォールトターゲットの許容範囲を実装している。安定性の研究成果であり、パフォーマンスをサポートする。任意の動作 Cully et systems al.[5]は、内部代替仮想マシンアプローチの実行と、フォールトトレラントフォールトトレランスVMのサポートとその実装のためのサポート(projectでは、多くのRemusと呼ばれる)を提案している。より高いレベルでは、この安定性のアプローチと性能。プリーの状態

cully VM et al. repeatally [5]は、フォールトトレラントなバックアップサーバーに移植するsupand sentのアプローチ実行中の代替案をチェックポイントで記述し、その実装をインフォマレーションでチェックポイントに収集している。チェックポイントと呼ばれる。チェックポイントは、チェックポイントが送信され、inforacknowledgedである。チェックポイントが収集された次のサーバーがバックアップされるまで、出力実行中の外部から送信され、遅延されなければならないのでチェックポイントがチェックポイントされる。ーションである。チェックポイントの利点は、この実行されたアプローチが頻繁に適用されることである(多くの場合、1秒あたりのウェルが等しく)。be 課題aまでの遅延主題は、このチェックポイント・アプローチが非常に高い送電ネットワークと帯域幅の知識を持っていることに従う。このアプローチのインクリメンタルな利点は、メモリに適合するように変更することで、各ユニプロセッサチェックポイントに等しくステートする。とRemus VMのマルチプロセッサの結果。5]の主な問題は、この225%への100%アプローチが、ネットワークカーネルコンパイルの帯域幅と要件SPECwebベンチマークに対して、インクリメンタルに変更を試みるときに、各2番目のチェックポイントごとに40のメモリチェックポイント状態に送出するために非常に高い速度低下を持つことを示すことである。1Gbit/sを使用した。[5]で送信のために提示されたコネクティブレムションの結果ネットワークは、225%のメモリ・スローダウン状態への100%の変更を示している。カーネルにはコンパイル数とSPECweb最適化ベンチマークがあり、1Gbit/sを使用してネットワーク2番目の帯域幅あたり40個のチェックポイントをデクレストで実行しようとすると有用であるが、ネットワークは明確なコネクションではない。1Gbit/sが接続されている。a 1 primary Gbit/sと接続の間に、20Mbit/s以下の帯域幅しか達成できないことは、多くの最適化とは対照的に、我々のアプローチは、ネットワークが10%未満の帯域幅を達成することができるが、オーバーヘッドが発生することを決定する際に有用である。一方、いくつかの実際のアプローチでは、バックアップホストを使用する。決定論的再生に基づくと、10%以下のオーバーヘッドを達成することができ、結論20Mbit/sの帯域幅と、WORKとWeのバックアップの間の必要なFUTUREは、ホストを設計し、いくつかの実際のアプリケーションを実装した。効率的でコム

7. VMware vSphereのシステムで、仮想マシンのフォールトコンクルージョン(FT)を提供し、将来的にWORKサーバー上で動作する。

クラスターである。我々は、VMwareの主要なシステムVMを、バックアップvSphere VMを介して効率的に実行し、構成することを基本として、仮想リプレイのために決定論的なtolVMware erance (FT)を使用してホスト障害を提供する。machines 実行中のサーバーがサーバー上で実行している場合、インプリアマリークラスタ。VM 我々の失敗、バックアップは、一次VMの中断がない状態で、またはバックアップデータの損失aを介して、実行を直ちに複製するベースのVMを設計する。VMwareを使用した別のホストでのVM性能の再生は、全体として決定論的である。フォールトトレラント VMpriware mary FT VM on で動作するサーバ VM が失敗した場合、バックアップハードウェア VM が取るコモディティは、すぐに優れ、10%のオーバーヘッドやデータの損失よりも少ない中断回数で表示される。いくつかの典型的なアプリケーション。ほとんどの場合
全体として、VMwareフォールトトレラントFTの性能コストは、VMから得られるVMである。FTは、汎用VMwareハードウェア決定論的を使用する上で、優れたオーバーヘッドウェアであり、リプレイして、同期している一部のVMに対して、10%未満のバックアップオーバーヘッドを示す。典型的なアプリケーション。低オーバーヘッド VMwareの性能FTのほとんどは、VMwareのコストは、VMwareから得られる効率FTから得られるリプレイを使用する決定論的オーバーヘッドに由来する。VMware さらに、必要なプライマリ VM を保持し、プライマリ VM をバックアップして同期させるために、ロギング再生帯域幅を決定論的に設定する。rebackback 低同期オーバーヘッドは通常VMwareが非常に小さいため、FTはMbit/sから20より導出が少ないことが多い。VMwareはロギングの決定論的な帯域幅再生を行うため、効率が良い。さらに、ほとんどのロギングケースで帯域幅が小さく、主要な構成とバックアップを維持することが望ましいと思われる。シンクプライマリでは、通常、バックアップが非常に小さく、VMの間隔が20 long Mbit/s未満であることが多い。距離は(1-100キロメートル)である。このように、VMware は FT が非常に小さいため、ほとんどのケースで使用可能である。シナリオでは、プライマリ VM とバックアップ VM が長い距離(1～100km)で区切られた構成を実装することは可能であると思われる。

to decrease the network transmission rate as the round-trip latency increases.

Table 2 gives our results for a number of measurements made by the standard `netperf` benchmark. In all these measurements, the client VM and primary VM are connected via a 1 Gbit/s network. The first two rows give send and receive performance when the primary and backup hosts are connected by a 1 Gbit/s logging channel. The third and fourth rows give the send and receive performance when the primary and backup servers are connected by a 10 Gbit/s logging channel, which not only has higher bandwidth, but also lower latency than the 1 Gbit/s network. As a rough measure, the ping time between hypervisors for the 1 Gbit/s connection is about 150 microseconds, while the ping time for a 10 Gbit/s connection is about 90 microseconds.

When FT is not enabled, the primary VM can achieve close (940 Mbit/s) to the 1 Gbit/s line rate for transmits and receives. When FT is enabled for receive workloads, the logging bandwidth is very large, since all incoming network packets must be sent on the logging channel. The logging channel can therefore become a bottleneck, as shown for the results for the 1 Gbit/s logging network. The effect is much less for the 10 Gbit/s logging network. When FT is enabled for transmit workloads, the data of the transmitted packets is not logged, but network interrupts must still be logged. The logging bandwidth is much lower, so the achievable network transmit bandwidths are higher than the network receive bandwidths. Overall, we see that FT can limit network bandwidths significantly at very high transmit and receive rates, but high absolute rates are still achievable.

6. RELATED WORK

Bressoud and Schneider [3] described the initial idea of implementing fault tolerance for virtual machines via software contained completely at the hypervisor level. They demonstrated the feasibility of keeping a backup virtual machine in sync with a primary virtual machine via a prototype for servers with HP PA-RISC processors. However, due to limitations of the PA-RISC architecture, they could not implement fully secure, isolated virtual machines. Also, they did not implement any method of failure detection or attempt to address any of the practical issues described in Section 3. More importantly, they imposed a number of constraints on their FT protocol that were unnecessary. First, they imposed a notion of epochs, where asynchronous events are delayed until the end of a set interval. The notion of an epoch is unnecessary – they may have imposed it because they could not replay individual asynchronous events efficiently enough. Second, they required that the primary VM stop execution essentially until the backup has received and acknowledged all previous log entries. However, only the output itself (such as a network packet) must be delayed – the primary VM itself may continue executing.

Bressoud [4] describes a system that implements fault tolerance in the operating system (Unixware), and therefore provides fault tolerance for all applications that run on that operating system. The system call interface becomes the set of operations that must be replicated deterministically. This work has similar limitations and design choices as the hypervisor-based work.

Napper et al. [9] and Friedman and Kama [7] describe implementations of fault-tolerant Java virtual machines. They follow a similar design to ours in sending information about inputs and non-deterministic operations on a logging channel. Like Bressoud, they do not appear to focus on detecting failure and re-establishing fault tolerance after a failure. In addition, their implementation is limited to providing fault tolerance for applications that run in a Java virtual machine. These systems attempt to deal with issues of multi-threaded Java applications, but require either that all data is correctly protected by locks or enforce a serialization on access to shared memory.

Dunlap et al. [6] describe an implementation of deterministic replay targeted towards debugging application software on a paravirtualized system. Our work supports arbitrary operating systems running inside virtual machines and implements fault tolerance support for these VMs, which requires much higher levels of stability and performance.

Cully et al. [5] describe an alternative approach for supporting fault-tolerant VMs and its implementation in a project called Remus. With this approach, the state of a primary VM is repeatedly checkpointed during execution and sent to a backup server, which collects the checkpoint information. The checkpoints must be executed very frequently (many times per second), since external outputs must be delayed until a following checkpoint has been sent and acknowledged. The advantage of this approach is that it applies equally well to uni-processor and multi-processor VMs. The main issue is that this approach has very high network bandwidth requirements to send the incremental changes to memory state at each checkpoint. The results for Remus presented in [5] show 100% to 225% slowdown for kernel compile and SPECweb benchmarks, when attempting to do 40 checkpoints per second using a 1 Gbit/s network connection for transmitting changes in memory state. There are a number of optimizations that may be useful in decreasing the required network bandwidth, but it is not clear that reasonable performance can be achieved with a 1 Gbit/s connection. In contrast, our approach based on deterministic replay can achieve less than 10% overhead, with less than 20 Mbit/s bandwidth required between the primary and backup hosts for several real applications.

7. CONCLUSION AND FUTURE WORK

We have designed and implemented an efficient and complete system in VMware vSphere that provides fault tolerance (FT) for virtual machines running on servers in a cluster. Our design is based on replicating the execution of a primary VM via a backup VM on another host using VMware deterministic replay. If the server running the primary VM fails, the backup VM takes over immediately with no interruption or loss of data.

Overall, the performance of fault-tolerant VMs under VMware FT on commodity hardware is excellent, and shows less than 10% overhead for some typical applications. Most of the performance cost of VMware FT comes from the overhead of using VMware deterministic replay to keep the primary and backup VMs in sync. The low overhead of VMware FT therefore derives from the efficiency of VMware deterministic replay. In addition, the logging bandwidth required to keep the primary and backup in sync is typically quite small, often less than 20 Mbit/s. Because the logging bandwidth is quite small in most cases, it seems feasible to implement configurations where the primary and backup VMs are separated by long distances (1-100 kilometers). Thus, VMware FT could be used in scenarios that

このように、VMware FTは、ロギングによって、小さなサイト全体が故障する災害から帯域幅が大幅に保護されるシナリオで使用できる。オーバーヘッドにCPUをかける価値がある。ログストリームは一般的にかなり圧縮可能であることに注意。我々の結果と単純なVMware圧縮FTでは、フォールトトレラントな大幅なVMのロギング帯域幅を減少させる効率的な実装が可能であることが示された技術があり、非定型的な余分なCPU再生時に小さなビルド量になる可能性がある。オーバーヘッド。このようなシステムは、透明性をもって

フォールト FTを実行するVMs VMwareの耐性結果は、システムが効率的で、フォールトトレラントなオーバーヘッドを最小限に抑えたアプリケーション実装であることを示した。VMs しかし、終端フォールトトレラント再生システム上に構築することができる。VMs 透明な顧客にとって有用なシステムであるため、VMs の実行や自動運転に非常に適した、堅牢で使いやすいシステムでなければならない。システム Aの使用可能でシステムアプリケーションは、他の多くの最小限のオーバーヘッドを必要とする。しかし、それ以上に、VMのフォールトトレラントなシステム実行を再現することができた。VM 特に、顧客にとって有用なVMwareであるためには、FTは自動的に、堅牢で冗長性が高く、故障後も、高度に自動化されたものを見つけ、それを復元しなければならない。適切な使用可能なサーバーシステムは、ローカルの多数のクラスタ他とコンポーネントが、その実行サーバー上で新しいバックアップを超えた複製VMを作成することを要求する。特に、VMwareに必要なFTの問題点、自動的に、その障害後のシステムの冗長性を回復することを実証し、ローカルデータセンターでサーバー顧客に適したアプリケーションを実用的に見つけることによって、使用可能である。クラスタリングし、その1つのサーバーに新しいバックアップVMを作成する。トレードオフの必要なフォールトトレランスの問題をすべて実装することで、我々は決定論的な実証再生システムを介して、現在実際のアプリケーションに使用可能な決定論的な再生が、顧客の実装済みデータセンターでのみ使用されていることを実証した。ユニプロセッサVMに効率的に適用できる。

しかし、VMを実装したユニプロセッサは、作業負荷の終端的な多種多様な再生を行うことで、許容範囲よりも故障が大きいというトレードオフがある。物理的な再生プロハサスのみが実装されているため、現在特に決定論的であり、強力に効率的に動作する。ユニプロセッサー VM。しかし、ワークロードのユニプロセッサは、多くのユニプロセッサよりも多くのユニプロセッサを使用することができ、その代わりに、ワークロードのスケーリングの多種多様な、特にマルチプロセッサ物理プロVM以来の1つの大きなものを使用することによって、VMを拡張することができる。cessors 高性能は、よりマルチプロセッサの性能を発揮するために、継続的に再生を獲得する。VMsでは、多くのワークロードが活発に研究されており、マイクロプロセッサでの使用によるスケーリングサポートのハードウェアを追加する代わりに、いくつかのVMで多くのユニプロセッサを使用することで、スケーリングが可能になり、潜在的にアウトすることができる。より大きなマルチプロセッサー 1つの興味深いVM。マルチプロセッサートランザクションVMのメモリを拡張するために、高性能な方向性がリプレイされるかもしれない。ハードウェアもマイクロプロセッサに興味を持っている。インターシステム拡張の方向性は、障害を拡張するためのハードウェアの一部である可能性がある。トランザクション部分メモリハードウェアモデルの失敗により、マルチプロセッサーを容易にするために、部分損失再生を意味する。機能性または再

dundancy 将来的には、サーバー内でも破損の拡大やデータの損失システムに興味がない。tem to 部分的なハードウェアを使ったディールの例は、損失の失敗である。ネットワークの部分的な接続性が故障した場合、VMにとって、機能的な電源の冗長な損失、またはサーバーサーバーの物理の再保証の部分的な損失を意味する。Ifがハードウェア破損の失敗やonデータ損失を部分的に引き起こさない場合。a server 実行例は、VM、多くのすべてのネットワークケースの損失(ただし、すべてに接続されていない)、VM、有利、または電力バックアップ電源VMに冗長に失敗する損失である。もし部分的にハードウェアの障害がすぐに回復できるのであれば、主要なVMであるクリティカルVMのサーバー上でフルサービスが発生する。多くの場合、VMは(しかしすぐにすべてを動かすわけではない)サーバー上で障害を起こすことは、潜在的に有利な信頼性の低いものから外れてしまう。をすぐにバックアップVMに転送する。このようなフェイルオーバーは、重要なVMの完全なAcknowledgmentsサービスを直ちに回復し、VMが潜在的に信頼できないサーバーから素早く移動されることを保証することができる。を多く生み出したクリシュナ・ラジャに感謝したい。

に関与した謝辞を、パフォーマンスの ts

Krishna replay, Raja, (生成された人を含む)に決定論的に感謝したい。デバイス) VMware Slava Malyugin, FTのLan実装Huang, Eric Loweに含まれる多数のFT機能者が参加した。コア・アレックス・イムミゴロドスキー、カウストゥブ決定論的パティルの補充者、リプレイ、ボリス(ワイズマンを含む、ペトル・ヴァンフォ・ドリヴンクをサポート、様々な、そしてミン・ヴァーチャル・シュー。また、ファンクショナル・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクションのファンクション・オア・ファンクション・オア・ファンクションのファンクション・オア・ファンクション・オア・ファンクション・オア・ファンクション・オア・ファンクションのファンクション・オア・ファンクションKaustubh Karyn Patil, Ritter Borisはワイズマンで優れたPetrの仕事を行った。を研究している。また、8.VMware REFERENCES vCenter の FT の上位管理には、他にも多くの人が関与している。カリン・リッターは、P.の仕事の多くであるアルスバーグを管理していた。とデイ、J.レジリエントのための原則

8. 参考文献第2回国際会議

[1] エンジニアリング・アルスバーグ、P.、(1976)、デイ、pp.627-644. J. 分散型リソースのレジリエントな共有のための原則.第2回ソフトウェア工学国際会議(1976)予稿集, pp.627-644.

[2] AMDフォールトトレランス. 株式会社SOSPのAMD64アーキテクチャ 15 *Programmer's Manual.* Sunnyvale, CA.

[3] Bressoud,T.、T.C.、TFT: Schneider, A Software F. システム・ハイパーバイザー・ベースの故障アプリケーション透過公差。フォールトトレランスの議事録にて。SOSP 15 In (Dec. 1995). 第28回年次大会講演論文集

[4] International Bressoud, T. Symposium C. TFT: ソフトウェアフォールトトレランスシステムによるアプリケーションの計算-透明化(1998年6月), pp.故障 128-137. 許容度. フォールト・トレランスAに関する第20回 G., Meyer, Annual D., Feeley, International Hutchison, N., Symposium and WarfieldのCully, B., Lefebvre論文集:128-137. 仮想マシン

[5] レプリケーション Cully, B., Lefebvre, Proceedings G., of Meyer, the Fifth D., USENIX Feeley, M., Symposium Hutchison, on N., Networked and Warfield, Systems A. Design Remus: and High Implementation Availability via Asynchronous (Apr.2008), pp. 仮想161-174. 機械の複製。Dunlap, G. In W., Proceedings King, S. of T., the Cinar, Fifth S., USENIX Basrai, Symposium M., and Chen, on Networked P. M. ReVirt: を通して設計侵入と実装分析を可能にするシステム。Virtual 2008), 機械 pp.161-174. ロギングと

[6] リプレイ。ダンラップ、S. the T., 2002 Cinar, Symposium S., Basrai, on Operating M., and Chen, Systems P. Design M. ReVirt: and Implementation Enabling Intrusion (Dec.2002). 仮想マシンのロギングとリプレイによる解析。Friedman, Proceedings R., and Kama, of the A. 2002 Transparent Symposium on Operating Fault-Tolerant Systems Java Design Virtual and Machine.実装 Proceedings (Dec.の2002年)。信頼性の高い分散システム(Oct.2003), pp.

[7] フリードマン、インテル社 R., and Kama, IntelA^ * A. R 64 透明でIA-32耐故障アーキテクチャソフトウェアJava仮想開発者のマシン。マニュアル。カリフォルニア州、リライアブルクララのサンタ議事録にて。分散システム(2003年10月)、pp.319-328.

[8] インテル・ナッパー社 J., Alvisi, L., Intêl and A^ * R Vin, 64 and H. A IA-32 Architectures フォールトトレラントソフトウェアJava仮想開発者マシン.マニュアル。国際カリフォルニア州サンタクララ議事録にて。ディペンダブル会議

[9] システムズ・ナッパー、J.、ネットワークス・アルビジ、L.、(2002年6月と)、Vin, H. pp.A 425-434. フォールトトレラント Nelson, M., Lim, Java B.-H., 仮想と機械.Hutchins, Proceedings G. Fast of Transparent 国際機械移行会議.2005年(2002年6月)、USENIX pp.のシステム会議およびネットワーク会議に依存する。425-434. 技術

[10] ネルソン、M.、(2005年4月)。B.-H.、Hutchins, G. 高速透過ナイチンゲール、マイグレーションE.B.、Veeraraghavan、仮想マシン用。K.、Chen, Proceedings P. M.、Flinn, 2005 J. Rethink Annual the USENIX Sync.テクニカル・イン・カンファレンス・プロシーディングス(4月の2005年)。2006年オペレーティングシンポジウム

[11] システムズ・ナイチンゲール、デザインE.とB.、実装ヴィーララガバン、(11月)。K., 2006)。陳、P.シュライティング、M.、フリン、R.、J.シュナイダー、F.シンクを再考する。B. プロシーディングスプロセッサにおけるフェイルストップ: フォールトトレラントなシステムコンピューティング設計システムの設計へのアプローチ2006シンポジウムの1つ。と実装 ACM コンピューティング (Nov.調査 2006)。1, 3

[12] (Aug. Schlicting, 1983), 222-238. R., and Schneider, F. B. フェイルストッププロセッサ: システムを用いたフォールトトレランス耐故障性コンピューティングサービスの設計への実装。状態ACMマシンコンピューティングアプローチ: サーベイズ Aチュートリアル1, 3 (Aug. ACM Computing 1983), 222-238. 調査22, 4(12月1990), 299-319.

[13] シュナイダー、ストラタス・テクノロジーズF. B. Stratusサービスからのベネフィット耐障害性の実装状態処理技術:マシンアプローチによる継続: 自動チュートリアル。ACM 99.999% マイクロソフト 22, 4 Windows の調査における稼働率計算 (Dec. 1990)、サーバー299-319.

[14] 環境。ストラタス・テクノロジーズhttp://www.stratus.com/ ~/media/Benefit from Stratus Stratus/Files/Resources/WhitePapers/continuous Continuing Processing Technology: マイクロソフトの6月Windowsサーバー2009の99.999%稼働率。環境http://www.stratus.com/ ~/media/Stratus/Files/Resources/WhitePapers/continuousXu, M., Malyugin, V., Sheldon, J., processing-for-windows.pdf, Venkitachalam, G., and Weissman, June B. ReTrace: 2009. 仮想マシンによる実行トレースの収集

[15] 決定論的シュー、M.、マリュウギン、リプレイ。V., Sheldon, J., of the 2007 Workshop Venkitachalam, on Modeling, G., and Benchmarking, Weissman, and B. ReTrace: シミュレーション・コレクタリング実行(2007年6月)。仮想マシン決定論的リプレイによるトレース。モデリング、ベンチマーク、シミュレーションに関する2007年ワークショップ(2007年6月)予稿集。

also protect against disasters in which entire sites fail. It is worthwhile to note that the log stream is typically quite compressible, and simple compression techniques can decrease the logging bandwidth significantly with a small amount of extra CPU overhead.

Our results with VMware FT have shown that an efficient implementation of fault-tolerant VMs can be built upon deterministic replay. Such a system can transparently provide fault tolerance for VMs running any operating systems and applications with minimal overhead. However, for a system of fault-tolerant VMs to be useful for customers, it must also be robust, easy-to-use, and highly automated. A usable system requires many other components beyond replicated execution of VMs. In particular, VMware FT automatically restores redundancy after a failure, by finding an appropriate server in the local cluster and creating a new backup VM on that server. By addressing all the necessary issues, we have demonstrated a system that is usable for real applications in customer's datacenters.

One of tradeoffs with implementing fault tolerance via deterministic replay is that currently deterministic replay has only been implemented efficiently for uni-processor VMs. However, uni-processors VMs are more than sufficient for a wide variety of workloads, especially since physical processors are continually getting more powerful. In addition, many workloads can be scaled out by using many uni-processor VMs instead of scaling up by using one larger multi-processor VM. High-performance replay for multi-processor VMs is an active area of research, and can potentially be enabled with some extra hardware support in microprocessors. One interesting direction might be to extend transactional memory models to facilitate multi-processor replay.

In the future, we are also interested in extending our system to deal with partial hardware failure. By partial hardware failure, we mean a partial loss of functionality or redundancy in a server that doesn't cause corruption or loss of data. An example would be the loss of all network connectivity to the VM, or the loss of a redundant power supply in the physical server. If a partial hardware failure occurs on a server running a primary VM, in many cases (but not all) it would be advantageous to fail over to the backup VM immediately. Such a failover could immediately restore full service for a critical VM, and ensure that the VM is quickly moved off of a potentially unreliable server.

Acknowledgments

We would like to thank Krishna Raja, who generated many of the performance results. There were numerous people involved in the implementation of VMware FT. Core implementors of deterministic replay, (including support for a variety of virtual devices) and the base FT functionality included Lan Huang, Eric Lowe, Slava Malyugin, Alex Mirgorodskiy, Kaustubh Patil, Boris Weissman, Petr Vandrovec, and Min Xu. In addition, there are many other people involved in the higher-level management of FT in VMware vCenter. Karyn Ritter did an excellent job managing much of the work.

8. REFERENCES

[1] Alsberg, P., and Day, J. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the Second International Conference on Software Engineering* (1976), pp. 627–644.

[2] AMD Corporation. *AMD64 Architecture Programmer's Manual*. Sunnyvale, CA.

[3] Bressoud, T., and Schneider, F. Hypervisor-based Fault Tolerance. In *Proceedings of SOSP 15* (Dec. 1995).

[4] Bressoud, T. C. TFT: A Software System for Application-Transparent Fault Tolerance. In *Proceedings of the Twenty-Eighth Annual International Symposium on Fault-Tolerance Computing* (June 1998), pp. 128–137.

[5] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchison, N., and Warfield, A. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proceedings of the Fifth USENIX Symposium on Networked Systems Design and Implementation* (Apr. 2008), pp. 161–174.

[6] Dunlap, G. W., King, S. T., Cinar, S., Basrai, M., and Chen, P. M. ReVirt: Enabling Intrusion Analysis through Virtual Machine Logging and Replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation* (Dec. 2002).

[7] Friedman, R., and Kama, A. Transparent Fault-Tolerant Java Virtual Machine. In *Proceedings of Reliable Distributed System* (Oct. 2003), pp. 319–328.

[8] Intel Corporation. *IntelÂ® 64 and IA-32 Architectures Software Developer's Manuals*. Santa Clara, CA.

[9] Napper, J., Alvisi, L., and Vin, H. A Fault-Tolerant Java Virtual Machine. In *Proceedings of the International Conference on Dependable Systems and Networks* (June 2002), pp. 425–434.

[10] Nelson, M., Lim, B.-H., and Hutchins, G. Fast Transparent Migration for Virtual Machines. In *Proceedings of the 2005 Annual USENIX Technical Conference* (Apr. 2005).

[11] Nightingale, E. B., Veeraraghavan, K., Chen, P. M., and Flinn, J. Rethink the Sync. In *Proceedings of the 2006 Symposium on Operating Systems Design and Implementation* (Nov. 2006).

[12] Schlicting, R., and Schneider, F. B. Fail-stop Processors: An Approach to Designing Fault-tolerant Computing Systems. *ACM Computing Surveys 1*, 3 (Aug. 1983), 222–238.

[13] Schneider, F. B. Implementing fault-tolerance services using the state machine approach: A tutorial. *ACM Computing Surveys 22*, 4 (Dec. 1990), 299–319.

[14] Stratus Technologies. Benefit from Stratus Continuing Processing Technology: Automatic 99.999% Uptime for Microsoft Windows Server Environments. At http://www.stratus.com/˜/media/-Stratus/Files/Resources/WhitePapers/continuous-processing-for-windows.pdf, June 2009.

[15] Xu, M., Malyugin, V., Sheldon, J., Venkitachalam, G., and Weissman, B. ReTrace: Collecting Execution Traces with Virtual Machine Deterministic Replay. In *Proceedings of the 2007 Workshop on Modeling, Benchmarking, and Simulation* (June 2007).