Fsas Technologies

P3AG-6612-02Z0

# ETERNUS AB series オールフラッシュアレイ

# ETERNUS AB series を使用した Microsoft SQL Server のベストプラクティスガイド

© 2022-2025 Fsas Technologies Inc.

# 目次

# 図目次

# 表目次

# はじめに

本書には、ストレージ管理者およびデータベース管理者が ETERNUS AB series ストレージに Microsoft SQL Server を正常にデプロイするために役立つ情報を掲載しています。

第 2 版
2025 年 3 月

## 登録商標

本製品に関連する他社商標については、以下のサイトを参照してください。
https://www.fujitsu.com/jp/products/computing/storage/trademark/
本書では、本文中の™、® などの記号は省略しています。

## 本書の読み方

### 対象読者

本書は、お客様の環境への SQL Server データベースソリューションを導入するお客様、パートナー、従業員、およびフィールドスタッフを対象としています。ソリューションのさまざまなコンポーネントに精通していることを前提としています。

### 関連マニュアル

ETERNUS AB に関連する最新の情報は、以下のサイトで公開されています。
https://www.fujitsu.com/jp/products/computing/storage/manual/

### 本書の表記について

#### 本文中の記号

本文中では、以下の記号を使用しています。

| | |
|---|---|
| **注意** | お使いになるときに注意していただきたいことを記述しています。必ずお読みください。 |
| **備考** | 本文を補足する内容や、参考情報を記述しています。 |

# 1. エグゼクティブサマリー

ほとんどの OLTP システムでは、サーバー内のプロセッサ、メモリ、および I/O サブシステムのバランスがよく、パフォーマンスのボトルネックとはみなされません。OLTP 環境におけるパフォーマンスの問題の主な原因は、通常、ストレージの I/O アクティビティに関連しています。既存の HDD ベースストレージシステムの速度は、サーバの処理能力と一致しません。

その結果、強力なプロセッサは頻繁にアイドル状態になり、ストレージ I/O 要求が完了するのを待ちます。この状況は、ユーザーとビジネスの生産性に悪影響を与えます。生産性が低下すると、ROI( 投資回収率 ) が低下し、TCO( 総保有コスト ) 全体が増加します。したがって、ストレージ IOPS のパフォーマンスとレイテンシは、ビジネスにおける戦略的な考慮事項となります。レスポンスタイムの目標が達成され、かつ他のシステムリソース ( プロセッサおよびメモリ ) のパフォーマンスの最適化が実現されていることを確認することが重要です。

ETERNUS AB series は、ミッションクリティカルなアプリケーションに対してミリ秒以下の卓越したレスポンスタイムを実現する堅牢なシステムです。ETERNUS AB series は、最新のソリッドステートドライブ (SSD) テクノロジーと多様なワークロードを処理してきた豊富な実績を活かして、レイテンシの影響を受けやすい高 I/O 環境を高速化し、優れたビジネス価値を提供します。

ETERNUS AB series は、OLTP データベースだけでなく、データウェアハウス環境にも非常に適しています。エントリーレベルの ETERNUS AB2100 は、Microsoft の Fast Track Data Warehouse プログラムで認定されています。

ETERNUS AB series は、高パフォーマンスと低レイテンシを提供することで市場をリードしています。AB6100 はコストパフォーマンスに優れたモデルで、AB5100 がこれに続きます。

NVMe は、Peripheral Component Interconnect Express (PCIe) SSD の業界標準インターフェースとなりました。合理化されたプロトコルコマンドセットと、I/O クロックサイクルの削減により、NVMe は、最大 64K キューとキューあたり最大 64K コマンドをサポートします。これらの属性により、SAS や SATA などの SCSI ベースのプロトコルよりも効率的になります。

NVMe over Fabrics(NVMe-oF) の導入は、インターフェースの特性である低レイテンシと小オーバヘッドに影響を与えることなく、NVMe のスケーラビリティを向上します。ETERNUS AB3100、AB5100、および AB6100 のシステムは、NVMe over RoCE(NVMe/RoCE)、NVMe over InfiniBand(NVMe/IB)、および NVMe over Fibre Channel(NVMe/FC) をサポートしています。

AB3100 と AB6100 の両方で、iSCSI および SCSI ベースのファイバチャネルプロトコル (FCP) と NVMe/FC をサポートします。NVMe/FC を実装するための規格変更はほとんど必要ないため、既存のストレージとともに NVMe/FC を簡単かつシームレスに導入することができ、既存ストレージにも影響をあたえません。NVMe/FC は他の FC トラフィックと同じインフラストラクチャコンポーネントを同時に使用できるため、組織に適したペースでワークロードを容易に移行できます。また、NVMe/FC を使用すると、NVMe コマンドと構造を変換せずにエンドツーエンドで効率的に転送できます。

エンドツーエンドの NVMe テクノロジーを基盤とする最先端の AB3100 と AB6100 は、サーバと SAN 間のレイテンシを削減するために NVMe-oF を提供するだけでなく、データへのアクセスを高速化するために NVMe テクノロジーを組み込んでいます。

ETERNUS AB series は、AB5100 では最大 1.8PB の未フォーマット SSD 容量、AB3100 および AB6100 では最大 367TB の容量を提供し、最も要求の厳しい組織の要件を満たす容量と堅牢な信頼性を提供します。本書では、ETERNUS AB series を使用した Microsoft SQL Server のベストプラクティスの概要について説明します。

## 1.1 注意事項

データベースを ETERNUS AB series ストレージシステムに再配置するか、ETERNUS AB series ストレージシステムで作成して高いパフォーマンスを実現することを前提としています。

また、OLTP アプリケーションのパフォーマンスを向上させることを前提としています。

# 2. ETERNUS AB series ストレージシステムの概要

## 2.1 ETERNUS AB series ハードウェアの概要

高い IOPS とミリ秒以下のレスポンスタイムを実現する ETERNUS AB series は、ビジネスクリティカルなアプリケーションの迅速な成果とカスタマーエクスペリエンスの向上を可能にします。

ETERNUS AB series は、高い IOPS と極めて低いレイテンシを兼ね備えており、高性能な専用ソリューションを必要とするデータベース駆動型アプリケーションに最適です。

表 2.1 では、ETERNUS AB series の性能の概要について説明します。

表 2.1 RAID6 における ETERNUS AB series の性能

| 性能比較 | AB2100 | AB3100 | AB5100 | AB6100 |
|---|---|---|---|---|
| SSD 数 | 24 | 24 | 48 | 24 |
| Read IOPS ( 最大 ) | 300K<210µs | 670K<250µs | 1M<250µs | 2M<250µs |
| Write IOPS ( 最大 ) | 45K<160µs | 100K<190µs | 185K<250µs | 340K<190µs |
| Read 帯域幅 | 10GB/ 秒 | 20GB/ 秒 | 21GB/ 秒 | 44GB/ 秒 |
| Write 帯域幅 ( キャッシュミラーリングが有効 ) | 3.7GB/ 秒 | 7GB/ 秒 | 9GB/ 秒 | 12.5GB/ 秒 |
| 低レイテンシ (Read) | 55K<140µs | 150K<200µs | 100K<120µs | 140K<110µs |
| 低レイテンシ (Write) | 45K<160µs | 50K<100µs | 150K<100µs | 200K<80µs |

表 2.2 は、ETERNUS AB series の各モデルの比較結果を示します。

表 2.2 ETERNUS AB series の各モデルの比較

| 製品比較 | AB2100 | AB3100 | AB5100 | AB6100 |
|---|---|---|---|---|
| 2U/24 拡張シェルフ | 3 | なし | 4 | なし |
| 総ドライブ数 | 96 | 24 | 120 | 24 |
| Raw 容量 | 1.45PB | 367TB | 1.8PB | 367TB |
| コントローラキャッシュオプション | 8GB または 32GB | 16GB | 16GB または 64GB | 32GB または 128GB |

性能とともに価値を最大化する鍵は、効率を最大化することです。これまで、企業は効率性を犠牲にして、ストレージのオーバープロビジョニングによって、極めて高いパフォーマンスレベルを達成してきました。しかし、状況は変わりつつあります。ETERNUS AB series システムは、オーバープロビジョニングを排除することでパフォーマンスと効率性のバランスを取るため、コストを大幅に削減できます。

従来型ドライブ 1,000 台以上のパフォーマンスを提供する ETERNUS AB series は、ラックスペース、電源、冷却装置を 95% 削減することで、厳しい要件にも対応できます。この機能は、部分的に搭載されたディスクを導入してアプリケーションのパフォーマンスを向上させることに慣れているお客様にとって大きなメリットとなります。

コスト効率に加えて、ETERNUS AB series では、アプリケーション効率も向上します。より多くのアプリケーション操作を完了することで、お客様は効率を大幅に向上でき、より良い結果を得ることができます。

ETERNUS AB series には、自動化されたフェイルオーバーを提供する完全冗長の I/O パスがあります。驚くべきことに自動化されたフェイルオーバーは今日利用可能なフラッシュ製品の多くでは提供されていないため、この技術を実装したい企業にとっては絶対的な要件となります。

すべての管理タスクは、ETERNUS AB series がオンライン状態で完全な Read/Write データアクセスを維持している間に実行されます。このアプローチにより、ストレージ管理者は、アプリケーションの I/O を中断することなく、構成の変更やメンテナンスを実行できます。

ETERNUS AB series は、データの損失やダウンタイムイベントから保護するために、エンタープライズストレージに共通の高度なデータ保護も提供します。この保護は、Snapshot テクノロジーを使用してローカルで利用でき、また、同期レプリケーションと非同期レプリケーションを使用してリモートからも利用できます（ただし、AB3100、AB6100 を除く）。

# 2.2 SANtricity OS

ETERNUS AB series は、ブラウザベースのアプリケーションの SANtricity System Manager によって管理されます。SANtricity System Manager はコントローラーに組み込まれています。

アレイ上にボリュームグループを作成する最初のステップとして、SANtricity の設定時に保護レベルを割り当てます。割り当てた保護レベルは、ボリュームグループを形成するために選択したディスクに適用されます。ETERNUS AB series は、Dynamic Disk Pool(DDP) と RAID レベル 0、1、5、6、および 10 をサポートします。DDP は、このマニュアルで説明するすべての設定に使用されています。

ストレージのプロビジョニングを簡素化するために、SANtricity には自動構成機能が用意されています。構成ウィザードは、アレイ上の使用可能なディスク容量を分析します。次に、容量要件、ホットスペア、およびウィザードで指定されたその他の条件を満たしながら、アレイのパフォーマンスとフォールトトレランスを最大化するディスクを選択します。

SANtricity Unified Manager および SANtricity System Manager の詳細については、マニュアルサイトを参照してください。

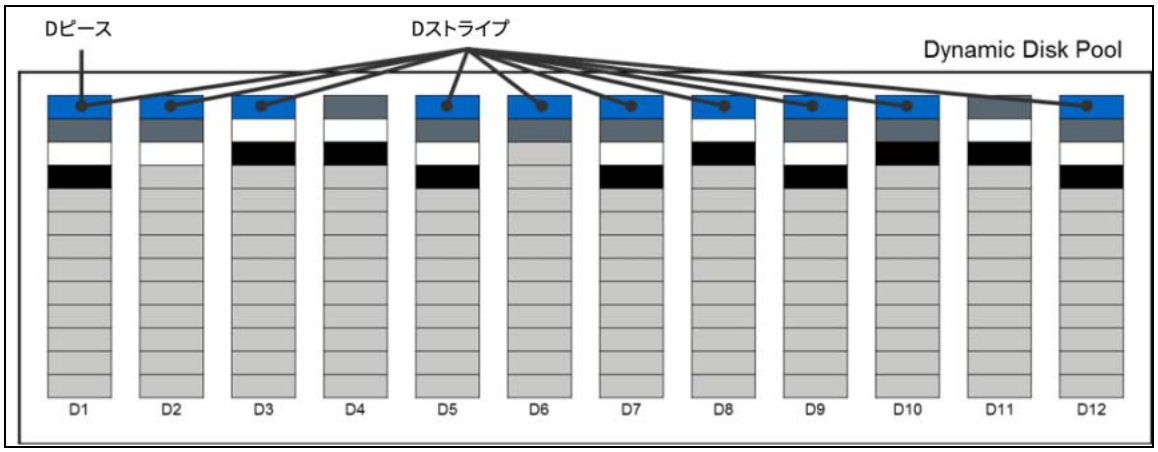
## 2.2.1 Dynamic Disk Pool

7 件の特許が出願中の DDP 機能は、データ、スペア容量、保護情報をドライブのプール全体に動的に分散します。これらのプールの範囲は、最小で 11 台のドライブから、最大でシステムがサポートするすべてのドライブまでです。ストレージ管理者は、単一のプールを作成するだけでなく、従来のボリュームグループと DDP または複数のプールを混在させることができ、これまでにないレベルの柔軟性を実現できます。

プールは、複数の下位レベルの要素で構成されます。最初の要素は D ピースです。D ピースは 512MB の一続きのセクションで、物理ドライブ内の 4,096 個の 128KB セグメントで構成されます。システムは、プール内の選択されたドライブから、インテリジェントな最適化アルゴリズムを使用して 10 個の D ピースを選択します。10 個の関連する D ピースを合わせると 1 つの D ストライプと見なされ、有効容量は 4GB になります。D ストライプの内容は RAID6(8+2) のシナリオと似ています。8 つの基本セグメントにユーザーデータ、1 つのセグメントにユーザーデータセグメントから計算されたパリティ (P) 情報、1 つのセグメントに RAID6 で定義された Q 値が含まれます。

その後、プール内の最大許容ボリュームサイズまで定義済みボリュームサイズを満たすために、必要に応じて複数の 4GB D ストライプアグリゲートからボリュームが作成されます。図 2.1 は、これらのデータ構造間の関係を示しています。

図 2.1 DDP のコンポーネント

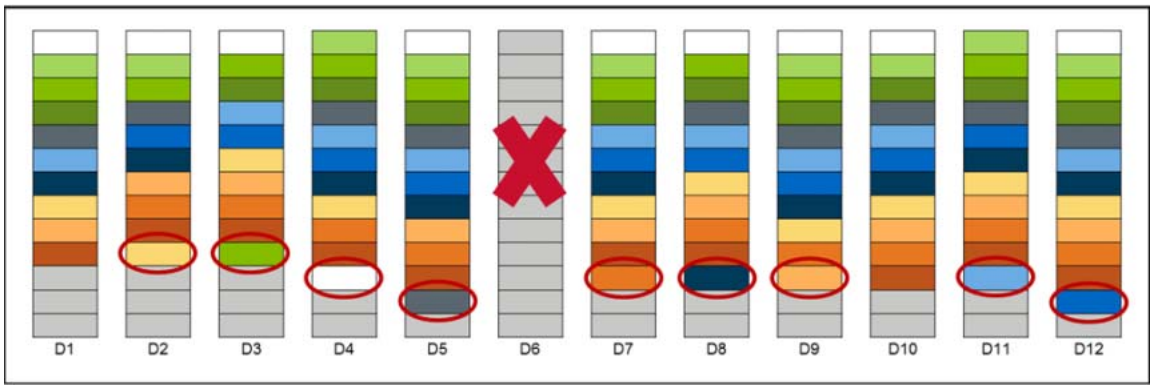



プールのもう 1 つの大きなメリットは、孤立状態の専用ホットスペアを使用するのではなく、プールに統合された保存容量があり、ドライブ障害に備えた再構築領域を提供できることです。このアプローチにより、個々のホットスペアを計画または管理する必要がなくなり、管理が合理化されます。また、従来のホットスペアとは対照的に、必要に応じて再構築の時間を大幅に改善し、再構築中のボリュームパフォーマンスを向上させます。

プール内のドライブに障害が発生すると、RAID6 で通常使用されるのと同じメカニズムを使用して、障害が発生したドライブの D ピースがプール内に潜在的に存在する他のすべてのドライブに再構築されます。このプロセスでは、コントローラフレームワーク内部のアルゴリズムによって、1 台のドライブに同じ D ストライプからの D ピースが 2 つ含まれていないことが検証されます。個々の D ピースは、選択したドライブ上で最も低い論理ブロックアクセス (LBA) 範囲で再構築されます。

図 2.2 は、ドライブ 6(D 6) に障害が発生した場合の例を示します。次に、そのディスク上に存在していた D ピースが、プール内の他の複数のドライブに同時に再作成されます。この処理には複数のディスクが関与しているため、この状況による全体的なパフォーマンスへの影響が軽減され、処理の完了に必要な時間が大幅に短縮されます。

図 2.2 DDP ドライブの障害




プール内で複数のディスク障害が発生した場合、データ可用性のリスクを最小限に抑えるために、D ピースが 2 つ欠落している D ストライプの再構築が優先されます。これらの重大な影響を受けた D ストライプが再構成された後、必要なデータの残りが再構成されます。

コントローラのリソース割り当ての観点から、DDP にはユーザーが変更可能な再構築優先順位 2 つ用意されています。

- Degraded reconstruction priority( デグレードの再構築優先度 ) は、影響を受けた D ストライプに再構築が必要な D ピースが 1 つしかない場合に適用される優先度です ( 値のデフォルトは High です )。
- Critical reconstruction priority( クリティカルな再構築優先度 ) は、D ストライプに再構築が必要な D ピースが 2 つある場合に適用される優先度です ( 値のデフォルトは Highest です )。

大規模なドライブプールで同時に 2 つのディスクに障害が発生した場合、2 つの D ピースの再構築が必要な状況に陥る D ストライプの数はそれほど多くはありません。前述したように、これらの重要な D ピースは、最初に最も高い優先順位で識別および再構築されます。そうすることで、プールはすぐにデグレード状態に戻り、それ以上のドライブ障害に耐えることができるようになります。

DDP は、再構築時間を短縮し、優れたデータ保護を提供するだけでなく、障害が発生した状態でのベースボリュームのパフォーマンスを、従来のボリュームグループのパフォーマンスと比べて大幅に向上させることができます。

DDP の詳細については、マニュアルサイトの「ETERNUS AB series、ETERNUS HB series SANtricity OS Dynamic Disk Pool 機能の説明とベストプラクティス」を参照してください。

## 2.3 ETERNUS AB series のプロビジョニング

SANtricity DDP テクノロジーは、ストレージ管理者に RAID 管理の合理化、データ保護の向上、あらゆる条件下での予測可能なパフォーマンスの維持を提供します。DDP は、ETERNUS AB series のドライブプール全体にデータ、保護情報、スペア容量を均等に分散し、セットアップを効率化して使用率を最大化します。この次世代テクノロジーは、ドライブ障害によるパフォーマンスへの影響を最小限に抑え、従来の RAID よりも最大 8 倍高速にシステムを最適な状態に戻すことができます。再構築時間が短縮され、再構築を優先する特許取得済みのテクノロジーにより、DDP は複数のディスク障害のリスクを大幅に軽減し、従来の RAID では実現できないレベルのデータ保護を提供します。

SANtricity では、ストレージをオンラインに保ったまますべての管理タスクを実行できるため、データへの常時アクセス ( 読み取り、書き込み ) が維持されます。ストレージ管理者は、接続されたホストへの I/O を中断することなく、構成の変更、メンテナンス、ストレージ容量の拡張を行うことができます。SANtricity のオンライン機能には、次のものがあります。

- 動的なボリューム拡張により、管理者は既存のボリュームの容量を拡張できます。
- 動的なセグメントサイズの移行により、管理者は特定のボリュームのセグメントサイズを変更できます。
- 動的な RAID レベルの移行では、データを再配置しなくても、既存のドライブ上の RAID グループの RAID レベルを変更できます。サポートされる RAID レベルは、0、1、5、6、および 10 です。
- 無停止でのコントローラファームウェアのアップグレードは、データアクセスを中断することなくサポートされます。

ETERNUS AB series のプロビジョニングの詳細については、マニュアルサイトを参照してください。ボリュームの作成とデプロイが繰り返される場合は、REST API コマンドを使用してこのタスクを自動化できます。

## ベストプラクティス

すべてのストレージ構成に Dynamic Disk Pool を使用することを推奨します。

# 3. SQL Server の概要

SQL Server データベースプラットフォームは、さまざまなアプリケーションをサポートできます。SQL Server を展開する前に、SQL Server インスタンスがサポートするアプリケーションのデータベースのワークロード要件を理解する必要があります。容量、パフォーマンス、可用性に関する要件はアプリケーションごとに異なるため、各データベースはこれらの要件を最適にサポートするように設計する必要があります。多くの組織では、アプリケーション要件を使用して SLA を定義し、データベースを複数の管理階層に分類しています。SQL Server データベースのワークロードは、次のように記述できます。

- OLTP データベースは、多くの場合、組織内で最も重要なデータベースです。これらのデータベースは、お客様向けのアプリケーションに対応しており、企業の主要業務に不可欠なものと考えられています。ミッションクリティカルな OLTP データベースとそれらがサポートするアプリケーションには、高レベルのパフォーマンスを必要とする SLA が設定されている場合がよくあります。これらのデータベースは、パフォーマンスの低下や可用性に影響を受けます。また、Windows フェイルオーバークラスタまたは Always On 可用性グループを使用したクラスタリングの候補になる場合もあります。これらのタイプのデータベースの I/O が混在すると、ランダム Read が 75% ～ 90%、Write が 25% ～ 10% という特徴があります。
- 意思決定支援システム (DSS) データベースは、データウェアハウスとも呼ばれます。これらのデータウェアハウスは、ビジネスをアナリティクスに依存している多くの組織にとってミッションクリティカルなものです。これらのデータベースは、クエリの実行時に CPU 使用率とディスクからの読み取り処理に影響されます。多くの組織では、DSS データベースが月末、四半期末、または年度末に最も重要なデータベースとなります。通常、このワークロードには 100% の読み取り I/O が混在しています。

## 3.1 SQL Server I/O の概要

SQL Server エンジンにはトランザクションを同時に処理する性質があるので、SQL Server は I/O レイテンシの影響を受けます。SQL Server は、行ロック、ページロック、エクステントロック、およびテーブルロックで構成される複雑なシステム上に構築され、SQL Server システム全体でトランザクションの一貫性を保ちます。I/O 構造がよくないと ( たとえば、I/O の応答に時間がかかりすぎる )、リソースが必要以上に長く保持され、システム内でブロッキングが発生します。ブロッキングが発生した場合、I/O サブシステムが根本原因であることはすぐには分からないことがほとんどです。

- **SQL Server の読み取り**
  SQL Server からデータを読み取る場合、クライアントは最初にバッファキャッシュにアクセスします。データがバッファキャッシュにない場合、SQL Server は I/O サブシステムにアクセスしてデータを取得します。データが 100% 読み取られるまでステートメントは完了しません。ユーザー接続またはプロセスは、完了するまで I/O 待機状態になります。
- **SQL Server 書き込み**
  ユーザーはトランザクションログとバッファキャッシュに書き込みます。変更するデータがまだバッファキャッシュにない場合は、I/O サブシステムからバッファキャッシュに読み込む必要があります。バッファマネジャーを使用すると、変更がデータベースに書き込まれる前に、トランザクションログを書き込むことができます。この手法はログ先行書き込み (WAL) と呼ばれています。ユーザーが変更を加えてコミットが実行されると、その変更に関するログ書き込みが表示され、コミットが完了します。コミットが完了すると、ユーザープロセスは変更がディスクに書き込まれるのを待たずに、次のステージまたはコマンドに進むことができます。ロールバックトランザクションはコミットと同じプロセスを逆の順番で実行します。バッファマネジャーはデータをキャッシュからディスクに移動します。各ログレコードのログシーケンス番号 (LSN) を追跡します。
- **トランザクションログ**
  SQL Server のトランザクションログは、シーケンシャルな書き込み中心の処理です。トランザクションログは、データベースまたはインスタンスに障害が発生した場合に、データリカバリに使用されます。

SQL Server 環境の OLTP データベースシステムは、最小限の時間で最大数のトランザクションをシステムで処理することに大きく依拠しています。異なるタイプの OLTP システムとして、ウェブ注文システムと製造追跡システムなどがあります。OLTP システムは、1 秒あたりのトランザクション数 (TPS) が大量になる場合があり、OLTP システムにとってはスループットがすべてです。これらのトランザクションを実行するために、SQL Server は効率的な I/O サブシステムを使用します。Microsoft SQL Server best practices article によると、OLTP トランザクションプロファイルには次の属性があります。

- OLTP 処理は、データファイルに対して発行される読み取りと書き込みの両方に対してランダムです。
- I/O アクティビティは、約 80% が読み取り、20% が書き込みです。
- 通常、読み取りアクティビティは一貫性があり、ポイントクエリを使用します。時間のかかる大きなクエリではありません。
- データファイルへの書き込みアクティビティは、チェックポイント処理中に発生します ( 頻度はリカバリ間隔の設定によって決まります )。
- ログの書き込みはシーケンシャルであり、ワークロードの性質に応じてサイズが異なります ( セクタを最大 60KB までアライメント )。
- ログの読み取りはシーケンシャルです ( セクタを最大 120KB までアライメント )。

## 3.2 Windows ボリュームマウントポイント

ETERNUS のストレージソリューションと Microsoft SQL Server はマウントポイントをサポートしています。マウントポイントは、別のボリュームをマウントするために使用できるボリューム上のディレクトリです。マウントされたボリュームにアクセスするには、マウントポイントのパスを参照します。マウントポイントを使用すると、Windows の 26 文字のドライブレター制限がなくなり、LUN 間でのデータ移動、ホスト間での LUN 移動、同一ホスト上での LUN のマウント解除やマウントをアプリケーション側で透過的に実行できます。これは、マウントポイントのパス名を変更せずに、基盤となるボリュームを移動できるためです。

### ベストプラクティス

Windows の 26 文字のドライブレター上限を超えるドライブ文字の代わりに NTFS マウントポイントを使用することを推奨します。ボリュームマウントポイントを使用する場合は、ボリュームラベルに指定する名前とマウントポイントに指定する名前が同じである必要があります。

## 3.3 データベースファイルとファイルグループ

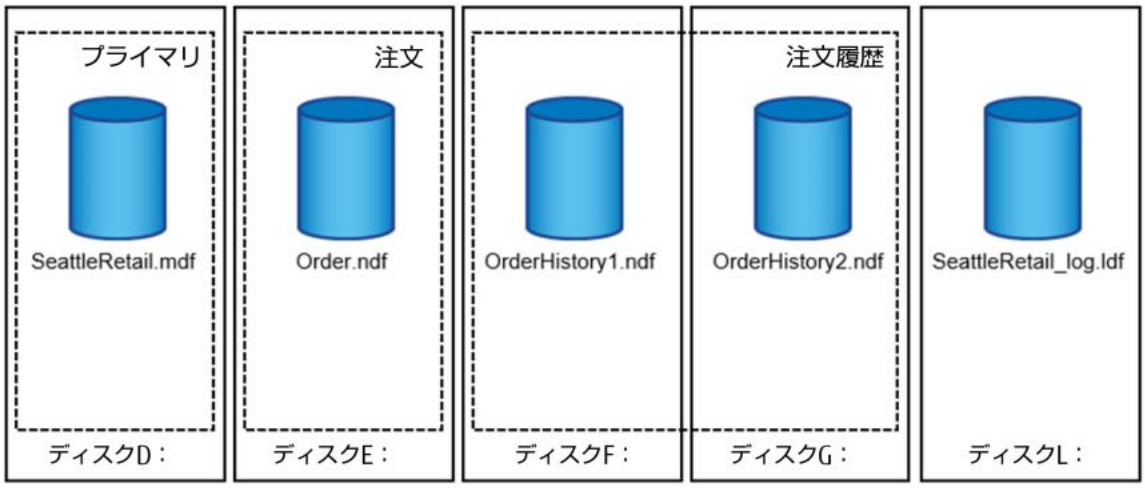
SQL Server データベースは、データの格納と操作を可能にするオブジェクトの集合です。理論的には、SQL Server (64 ビット ) はインスタンスあたり 32,767 のデータベースと 524,272TB のデータベース領域をサポートしますが、通常のインストールでは少数のデータベースしか使用されません。ただし、SQL Server が処理できるデータベースの数は、負荷とハードウェアによって異なります。SQL Server インスタンスが数十、数百、数千の小さなデータベースをホストすることは珍しくありません。

各データベースは、1 つ以上のデータファイルと 1 つ以上のトランザクションログファイルで構成されます。トランザクションログには、データベーストランザクションおよび各セッションで行われたすべてのデータ変更に関する情報が格納されます。データが変更されるたびに、SQL Server は、アクションのアンドゥ ( ロールバック ) またはリドゥ ( リプレイ ) を実行するために十分な情報をトランザクションログに保存します。SQL Server のトランザクションログは、データの整合性と堅牢性に対する SQL Server の高い評価の一部を担います。トランザクションログは、SQL Server の原子性、一貫性、分離性、および永続性 (ACID 特性 ) に不可欠です。データページが変更されると、SQL Server はトランザクションログに書き込みます。すべてのデータ操作言語 (DML) 文 ( たとえば、select 文、insert 文、update 文、または delete 文 ) は完全なトランザクションであり、トランザクションログはセットベースの処理全体が確実に実行されるようにします。このようにして、ログはトランザクションの原子性を確認します。

各データベースには 1 つのプライマリデータファイルがあり、デフォルトでは拡張子は .mdf です。また、各データベースには、デフォルトで .ndf 拡張子の付いたセカンダリデータベースファイルを含めることができます。図 3.1 を参照してください。

すべてのデータベースファイルはファイルグループにグループ化されます。ファイルグループは、データベース管理を簡素化する論理ユニットです。これにより、論理オブジェクトの配置と物理データベースファイルを分離できます。データベースオブジェクトテーブルを作成するときは、基礎となるデータファイル構成を考慮せずに、どのファイルグループに配置するかを指定します。

図 3.1　　各ドライブのプライマリ、セカンダリ、およびログファイルの例




ファイルグループ内に複数のデータファイルを配置できるため、異なるストレージデバイスに負荷を分散でき、システムの I/O パフォーマンスの向上に役立ちます。これとは対照的に、トランザクションログでは、SQL Server がトランザクションログにシーケンシャルに書き込むため、複数のファイルによるメリットはありません。

ファイルグループ内の論理オブジェクトの配置と物理データベースファイルの配置を分離することで、データベースファイルのレイアウトを微調整し、ストレージサブシステムを最大限に活用できます。たとえば、異なる顧客に製品を導入する独立系ソフトウェアベンダー (ISV) は、基礎となる I/O 構成および導入段階で予想されるデータ量に応じてデータベースファイルの数を調整できます。これらの変更は、データベースファイルではなくファイルグループにデータベースオブジェクトを配置するアプリケーション開発者にとっては透過的です。

Microsoft では、システムオブジェクト以外にプライマリファイルグループを使用しないことを推奨しています。ユーザーオブジェクト用に個別のファイルグループまたはファイルグループのセットを作成すると、特にデータベースが大きい場合に、データベース管理とディザスタリカバリが容易になります。

データベースの作成時または既存データベースへの新規ファイルの追加時に、初期ファイルサイズおよび自動拡張パラメータを指定できます。SQL Server は、プロポーショナルフィルアルゴリズムを使用して、データを書き込むデータファイルを選択します。ファイルで使用可能な空き領域に比例してデータ量が書き込まれます。ファイル内の空き領域が多いほど、処理される書き込み量も多くなります。

## ベストプラクティス

単一ファイルグループ内のすべてのファイルに同じ初期サイズと自動拡張パラメータを設定し、拡張サイズをパーセントではなくメガバイト単位で定義することを推奨します。このアプローチは、プロポーショナルフィルアルゴリズムがデータファイル間で書き込みアクティビティを均等に分散するのに役立ちます。

# 3.4 SQL Server データベースファイル

ETERNUS AB series にデータベースファイルをプロビジョニングするには、次の 2 つの方法があります。

- ETERNUS AB series の LUN に存在するデータベースファイルを含むデータベースを作成するには、作成時に次の Transact-SQL (T-SQL) スクリプトを使用できます。

```
-- Assuming C:\MSSQL\Data and C:\MSSQL\Log is the mount points of EF-Series LUNs
USE master;
GO
CREATE DATABASE Sales
ON
( NAME = Sales dat,
 FILENAME =
 'C:\MSSQL\Data\saledat.mdf', SIZE =
 10,
 MAXSIZE = 50,
 FILEGROWTH =
 5 )
LOG ON
( NAME = Sales_log,
 FILENAME =
 'C:\MSSQL\Log\salelog.ldf', SIZE =
 5MB,
 MAXSIZE = 25MB,
 FILEGROWTH =
 5MB ) ;
GO
```

- データベースファイルを ETERNUS AB series 以外の LUN から ETERNUS AB series LUN に移動するには、データベースファイルをデタッチする必要があります。データベースファイルをデタッチした後、ETERNUS AB series LUN 上にあるパスまたはマウントポイントにファイルをコピーできます。ファイルがコピーされたら、新しい場所にデータベースファイルをアタッチできます。

一般的なベストプラクティスは、データ、トランザクションログ、tempdb ファイルを別々の論理 LUN に分けることです。この推奨事項は、異なるタイプのワークロードを別々の物理ストレージに配置するという考え方に基づいています。この推奨事項は、そのような分離が可能な環境で有効です。ただし、物理的に分離するのが非常に難しい共有ストレージ環境に SQL Server を導入している場合がほとんどです。このようなケースでは、通常はパフォーマンスの観点からも分離は必要でさえありません。

潜在的な問題を切り分けやすくするために、管理性を高めるために分離を維持することを推奨します。たとえば、tempdb を独自の論理ディスクに配置すると、ほかのファイルの領域要件を気にすることなく、ディスク容量いっぱいまで tempdb のサイズを事前に設定できます。分離して配置するファイルが多いほど、論理ディスクのパフォーマンスを特定のデータベースファイルに簡単に関連付けることができます。

# 3.5 tempdb のデータベースファイル

tempdb システムデータベースは、SQL Server インスタンスに接続しているすべてのユーザーが使用できるグローバルリソースであり、次のデータを格納するために使用されます。

- グローバルまたはローカルの一時テーブル、一時ストアドプロシージャ、テーブル変数、カーソルなど、明示的に作成される一時的なユーザーオブジェクト。
- スプールまたは並べ替えの中間結果を格納するワークテーブルなど、SQL Server データベースエンジンによって作成される内部オブジェクト。
- READ COMMITTED の行バージョン分離トランザクションまたはスナップショット分離トランザクションを使用するデータベース内のデータ変更トランザクションによって生成される行バージョン。
- オンラインのインデックス操作、複数のアクティブな結果セット (MARS) 、AFTER トリガーなど、機能のデータ変更トランザクションによって生成される行バージョン。

tempdb 内の処理は最小限のログで記録され、トランザクションをロールバックできます。tempdb は SQL Server が起動されるたびに再作成されるため、システムは空の tempdb データベースから起動します。一時テーブルとストアドプロシージャは切断時に自動的に削除され、システムのシャットダウン時にアクティブな接続はありません。したがって、SQL Server のセッション間で tempdb に保存するものはありません。tempdb では、バックアップおよびリストアは実行できません。

すべての SQL Server インスタンスには tempdb という名前の共有データベースがあり、一時オブジェクトによって使用されます。インスタンスごとに tempdb データベースが 1 つしかないため、このデータベースを頻繁に使用するシステムでボトルネックが発生することがよくあります。通常、このボトルネックは、データファイル内の割り当てビットマップページでのメモリ内ラッチの競合である PAGELATCH が原因で発生します。

同じ初期サイズと自動拡張構成のデータファイルを tempdb に追加することで、メモリ内ページの競合を削減できます。このようなアプローチが可能なのは、SQL Server では、ラウンドロビン方式のプロポーショナルフィルアルゴリズムを使用して、データファイル間で書き込みをストライピングするためです。データベースに複数のデータファイルが存在する場合、ファイルへのすべての書込みがそれら複数のファイルにストライピングされます。特定のファイルへの書き込みは、すべてのファイルの合計空き領域と比較した空き領域の割り合いに基づいて実行されます。したがって、書き込みは、ファイルサイズに関係なく、空き領域に応じて相対的に配分され、ファイルが同時にいっぱいになるようにします。

Microsoft では、ファイルと論理 CPU の間のマッピングを 1 対 1 にすることを推奨しています。膨大なワークロードでのテストでは、データファイルが数百個あってもパフォーマンス上のメリットを確認しています。

ただし、より実用的なアプローチでは、ファイルと論理 CPU の間に最大 8 つの 1 対 1 のマッピングを作成します。割り当て競合が引き続き発生する場合や、I/O サブシステムをより強化する必要がある場合は、ファイルを追加できます。

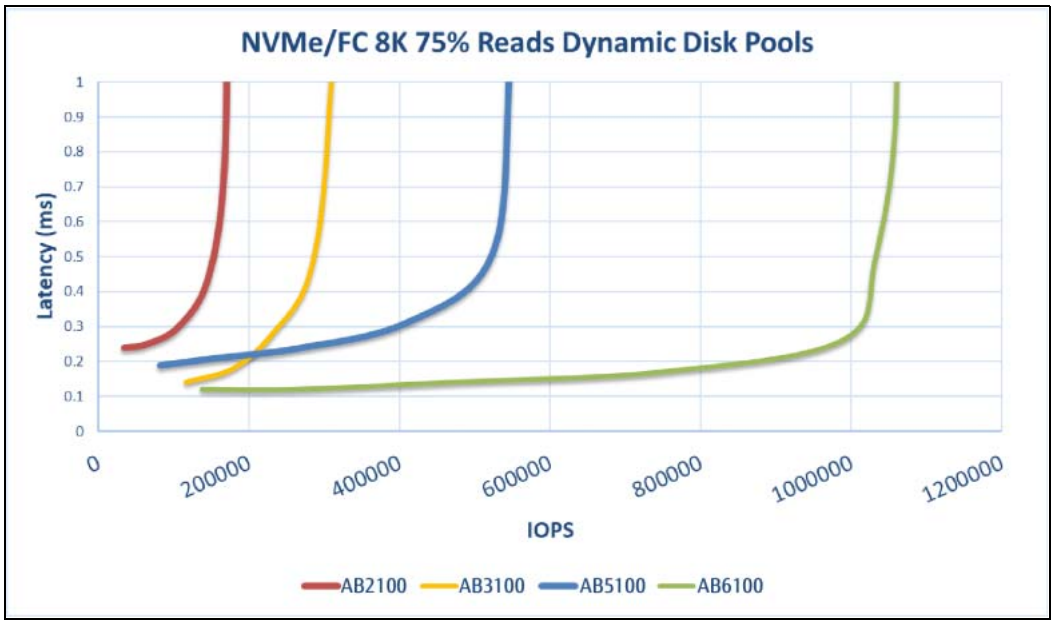
ETERNUS AB series は、卓越したパフォーマンスを実現する堅牢なシステムであるため、複数の tempdb ファイルを作成して、ETERNUS AB series に配置できます。これを行うには、次の T-SQL Server スクリプトを使用します。

```
select *
from    sys.database_files

use master
go
-- Change logical tempdb file name first since SQL Server shipped with logical file name called
tempdev
alter database tempdb modify file (name = 'tempdev', newname = 'tempdev01');

-- Change location of tempdev01 and log file to C:\MSSQL\Tempdb path
alter database tempdb modify file (name = 'tempdev01', filename =
'C:\MSSQL\Tempdb\tempdev01.mdf');
alter database tempdb modify file (name = 'templog', filename = 'C:\MSSQL\Tempdb\templog.ldf');

-- Assign proper size for tempdev01
ALTER DATABASE [tempdb] MODIFY FILE ( NAME = N'tempdev01', SIZE = 2GB, FILEGROWTH = 100 MB );
ALTER DATABASE [tempdb] MODIFY FILE ( NAME = N'templog', SIZE = 4GB, FILEGROWTH = 100 MB );

-- Add more tempdb files
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev02', FILENAME =
N'C:\MSSQL\Tempdb\tempdev02.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev03', FILENAME =
N'C:\MSSQL\Tempdb\tempdev03.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev04', FILENAME =
N'C:\MSSQL\Tempdb\tempdev04.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev05', FILENAME =
N'C:\MSSQL\Tempdb\tempdev05.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev06', FILENAME =
N'C:\MSSQL\Tempdb\tempdev06.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev07', FILENAME =
N'C:\MSSQL\Tempdb\tempdev07.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev08', FILENAME =
N'C:\MSSQL\Tempdb\tempdev08.ndf' , SIZE = 2GB , FILEGROWTH = 100 MB);
```

# 4. ETERNUS AB series の OLTP パフォーマンス

図 4.1 は、一般的な OLTP ワークロードにおいて DDP で構成した ETERNUS AB series( エントリーモデルの AB2100、ミッドレンジモデルの AB5100、エンドツーエンド NVMe モデルの AB3100 および AB6100) の性能結果を示します。試験は、プールで 24 台の SSD を使用し、AB3100 と AB6100 では FC ホストプロトコルまたは NVMe/FC を使用し、ワークロードを Read 75% と Write 25% に設定したアレイで実施されました。レイテンシが約 300µs のとき、AB2100 は 100,000IOPS、AB5100 は 400,000IOPS を実現しますが、NVMe ベースの AB3100 と AB6100 はそれぞれ 240,000IOPS と 1,000,000IOPS 以上を実現します。

図 4.1　　OLTP 構成のパフォーマンス結果の比較



# 5. 高可用性

## 5.1 ETERNUS AB series システムと SANtricity OS

ETERNUS AB series ストレージシステムは、高い信頼性と高可用性を実現するように設計されており、次のような機能を備えています。

- 自動 I/O パスフェイルオーバーを備えたデュアルアクティブコントローラ
- RAID レベル 0、1、5、6、10、または DDP
- ホットスワップ対応の冗長コントローラ、ディスク、電源装置、およびファン
- グローバルホットスペアを使用した自動ドライブフェイルオーバー検出と再構築
- バッテリのバックアップとメモリへのデステージ機能を備えたミラーデータキャッシュ
- 無停止でのコントローラファームウェアアップグレード
- ドライブの健全性を積極的に監視
- 自動パリティチェックと修正を備えたバックグラウンドメディアスキャン

すべてのコンポーネントは完全な冗長性を備えており、システムの電源を切ったり、動作を停止したりすることなく、コンポーネントを交換できます。冗長コンポーネントには、コントローラ、ディスク、電源装置、およびファンが含まれます。ETERNUS AB series の電源装置は、80-PLUS 認証電源です。ETERNUS AB series は、あらゆる状況でデータを保護するために設計されたいくつかの機能を備えています。さまざまな冗長性レベルで複数の RAID レベルを使用できます。接続が失われると、あるパスから別のパスへの自動フェイルオーバーもシステムに含まれます。シェルフ内では、各ドライブが各コントローラに接続されるため、内部接続の問題も迅速に解決できます。ETERNUS AB/HB series で作成したボリュームは、作成後すぐにホスト I/O に使用できます。また、I/O を中断することなく、多くのプロパティを変更できます。

データを保護する ETERNUS AB series のその他の機能には、コントローラキャッシュのミラーリングとバックアップがあります。システムの動作中に電源が失われた場合は、オンボードバッテリがキャッシュメモリから内部コントローラフラッシュにデータをデステージして、電源が復旧したときにデータを使用できるようにします。RAID アルゴリズムを使用すると、万が一ドライブに障害が発生した場合でも、失われたデータをシステムで再作成できます。また、RAID パリティを使用してデータを確認し、読み取り不能セクタにヒットした場合は再構築を続行することもできます。

システムは、データを常に保護する他のタスクをバックグラウンドで実行します。オプションのメディアスキャン機能では、現在どのホストからもアクセスされていないセクタについても、不整合を検索します。ETERNUS AB series は、SSD の消耗を積極的に追跡し、寿命が近づいているドライブにフラグを立てます。すべてのタイプの診断データは、必要に応じて当社サポート部門が後で使用できるように継続的に収集されます。

すでに説明したように、ETERNUS AB series には、信頼性と可用性のための様々な機能が用意されています。さらに、SANtricity を使用すると、可用性を最大限に高めることができます。たとえば、SANtricity には次の機能があります。

- 高速かつ効率性に優れた Snapshot テクノロジーを活用
- 数秒でデータを保護
- 変更されたブロックのみを格納することでフラッシュの消費量を削減
- 堅牢なディザスタリカバリ保護を提供
- 同期ミラーリングをサポートし、データ消失ゼロのコンテンツ保護を実現
- 非同期ミラーリングよって遠隔地の保護とコンプライアンスをサポート
- 柔軟な保護によって ROI を最大化
- コストとパフォーマンスのニーズに基づいて、フラッシュ、ニアライン SAS(NL-SAS)、または複数のリカバリターゲットの混在をサポート
- 予算を超えることなく迅速に処理

# 5.2 SQL Server HA オプション

SQL Server の高可用性ソリューションは、ハードウェア障害またはソフトウェア障害の影響を目立たなくし、アプリケーションの可用性を維持して、ユーザーが認識するダウンタイムを最小限に抑えます。SQL Server には、いくつかの高可用性 (HA) ソリューションが用意されています。

## 5.2.1 ログ配布

ログ配布はデータベースレベルで動作します。本番環境の単一のデータベース（プライマリデータベース）に対して、1 つ以上のウォームスタンバイデータベース ( セカンダリデータベース ) を維持できます。ログ配布の詳細については、ログ配布について (SQL Server) を参照してください。

## 5.2.2 データベースミラーリング

データベースミラーリングは、ほぼ瞬時のフェイルオーバーをサポートし、データベースの可用性を向上させます。データベースミラーリングでは、本番環境のデータベース（プリンシパルデータベース）に対して、単一のスタンバイデータベースまたはミラーデータベースを維持できます。詳細については、データベースミラーリング (SQL Server) を参照してください。

## 5.2.3 Always On フェイルオーバークラスタインスタンス

Always On フェイルオーバークラスタインスタンスは、Windows Server Failover Clustering(WSFC) 機能を利用して、サーバーインスタンスレベル ( フェイルオーバークラスタインスタンス (FCI)) の冗長性によってローカルの高可用性を提供します。FCI は、WSFC ノード間、場合によっては複数のサブネット間にインストールされる単一の SQL Server インスタンスです。ネットワーク上では、FCI は 1 台のコンピュータ上で実行されている SQL Server インスタンスのように見えますが、現在のノードが使用できなくなった場合、FCI は WSFC ノード間でフェイルオーバーを実施します。詳細については、Always On フェールオーバークラスターインスタンス (SQL Server) を参照してください。

## 5.2.4 Always On 可用性グループ

Always On 可用性グループは、SQL Server 2012 で導入されたエンタープライズレベルの高可用性およびディザスタリカバリソリューションで、1 つ以上のユーザーデータベースの可用性を最大化します。Always On 可用性グループでは、SQL Server インスタンスが WSFC ノード上に存在する必要があります。詳細については、Always On 可用性グループ (SQL Server) を参照してください。

Always On 可用性グループでは、非同期コミットモードと同期コミットモードの 2 つの可用性モードがサポートされています。

- 非同期コミットモードは、可用性レプリカが距離を隔てて分散されている場合に効果的に機能するディザスタリカバリソリューションです。詳細については、非同期コミット可用性モードを参照してください。
- 同期コミットモードでは、トランザクションのレイテンシが増加しますが、パフォーマンスよりも高可用性が重視されます。同期コミットモードでは、セカンダリレプリカがログをディスクに保存するまで、クライアントにトランザクション確認は送信されません。詳細については、同期コミット可用性モードを参照してください。

自動フェイルオーバーでは、プライマリレプリカの消失後にデータベースが素早く再使用可能になるため、高可用性が実現されます。自動フェイルオーバーのための可用性グループを構成するには、現在のプライマリレプリカと 1 つのセカンダリレプリカの両方に、自動フェイルオーバーを指定した同期コミットモードを設定する必要があります。

# 6. サイジング

SQL Server のパフォーマンスは I/O に重点が置かれてきました。従来、ユーザーはスピンドルの数を増やすか、スピンドルの回転速度を上げることで、このパフォーマンスを向上させていました。ETERNUS AB series の登場により、SSD を使用することでパフォーマンスを向上させることができるようになりました。

## 6.1 ETERNUS AB series I/O の概要

ETERNUS AB series ストレージシステムの全体的なパフォーマンスに影響を与える要因には、ネットワークインフラストラクチャなどの物理コンポーネントや、基盤となるストレージ自体の構成などがあります。一般的に、ストレージシステムのパフォーマンス調整には、40/30/30 ルールが適用されます。調整および設定の 40% はストレージシステムレベル、30% はファイルシステムレベル、30% はアプリケーションレベルで行うというものです。次の項では、ストレージシステムの仕様に関連する 40% について説明します。ファイルシステムおよびアプリケーションレベルの場合、一般的な考慮事項には次のものが含まれます。

- **I/O サイズ**
  ETERNUS AB series ストレージシステムは、主に応答性に優れたシステムです。I/O 操作を完了するには、ホストがその操作を要求する必要があります。ホストからの各要求の I/O サイズは、IOPS 数またはスループット（メガバイト / 秒 [MBps] またはギガバイト / 秒 [GBps]）に大きな影響を与える可能性があります。一般的に、I/O 操作が多いほど、IOPS 数は少なくなり、MBps も大きくなります。その逆も同様です。この関係は、スループット =IOPS×I/O サイズという計算式で定義されます。
- **読み取り要求と書き込み要求**
  I/O サイズに加えて、ストレージシステムレベルで処理される読み取り I/O 要求と書き込み I/O 要求の割合もストレージシステムに影響を与える可能性があります。ソリューションを設計するときは、この割合も考慮する必要があります。
- **シーケンシャルデータストリームとランダムデータストリーム**
  基盤となるディスクメディアの論理ブロックアドレスへのホストリクエストは、シーケンシャルまたはランダムに行うことができ、ストレージシステムレベルでのパフォーマンスに大きな影響を与えます。このシーケンスは、物理メディアが最小限のレイテンシで要求に効果的に応答できるかどうかに影響します。また、このシーケンスは、ストレージシステムのキャッシュアルゴリズムの有効性にも影響します。例外として、ランダム要求のレイテンシが長くならないのは、レイテンシが機械的に発生しない SSD の場合です。
- **同時 I/O 処理数**
  特定のボリュームに適用される未処理の I/O 操作の数は、ファイルシステムが raw I/O、バッファ I/O、または直接 I/O を使用するかどうかなど、いくつかの要因によって異なります。一般に、ETERNUS AB series ストレージシステム内のほとんどのボリュームは、複数のドライブでストライピングされています。個々のディスクに最小限の未処理 I/O を割り当てた場合、ストレージシステム内のリソースが十分に活用されず、必要とされるパフォーマンスを確保できない場合があります。

ETERNUS AB series テクノロジーを初めて使用する場合は、RAID10、RAID5、RAID6、および DDP テクノロジーの違いを確認することをお勧めします。表 6.1 は、さまざまな RAID レベルの有効容量を比較したものです。完全を期すために、ETERNUS AB series システムでサポートされているすべての RAID レベルを示しています。

表 6.1 RAID レベルごとの有効容量の比較

| 求められる機能 | RAID 0 | RAID 1<br>RAID 10 | RAID 5 | RAID 6 | DDP |
|---|---|---|---|---|---|
| 有効容量 | 100% | 50% | (N-1) / N (N はボリュームグループ内で選択されているドライブ数 ) | (N-2) / N (N はボリュームグループ内で選択されているドライブ数 ) | 80% から、選択されている予約済み容量を引いた容量 |

# 6.2 I/O の見積り

データベースのサイズを設定するときは、システムに必要な I/O 操作の数を見積もることが重要です。ここでは、データベースインスタンスのパフォーマンスを許容範囲内に維持する方法について説明します。システムの実際の物理 I/O 番号を取得できない場合は、I/O を見積もる必要があります。これは通常、新しいシステムを構築中に発生します。次の項では、I/O を見積もる計算式を示します。

## 6.2.1 新規の OLTP データベースシステム

次の項目を考慮する必要があります。

- ビジネストランザクション
- ビジネストランザクションの期間
- アプリケーションの許容可能なレイテンシ
- トランザクションとシステムトランザクション (I/O) のおおよその割合

次の例は、ETERNUS AB series で SQL Server のサイズを見積もる最も簡単な方法を示しています。

1. ビジネストランザクションの数を見積もります。たとえば、1 日あたりのビジネストランザクション数を、600,000,000 と見積もります。ビジネスは、24 時間 365 日体制で運営されます。
2. 1 つのビジネストランザクションで 25 個の I/O 操作が作成されるとします。したがって、この例では、1 日あたりの I/O 操作は、600,000,000 x 25 = 15,000,000,000　I/O です。
3. 標準的なストレージサイジングを使用するには、IOPS 情報が必要です。この例では、(15,000,000,000) / 86,400 = 173,612 IOPS です。
4. アプリケーションの許容可能なレイテンシを考慮する必要があります。たとえば、アプリケーションは 1 ミリ秒未満のレイテンシを必要とする場合があります。
5. Read と Write の割合も見積もる必要があります。OLTP データベースでは、一般的に Read が 80%、Write が 20% です。

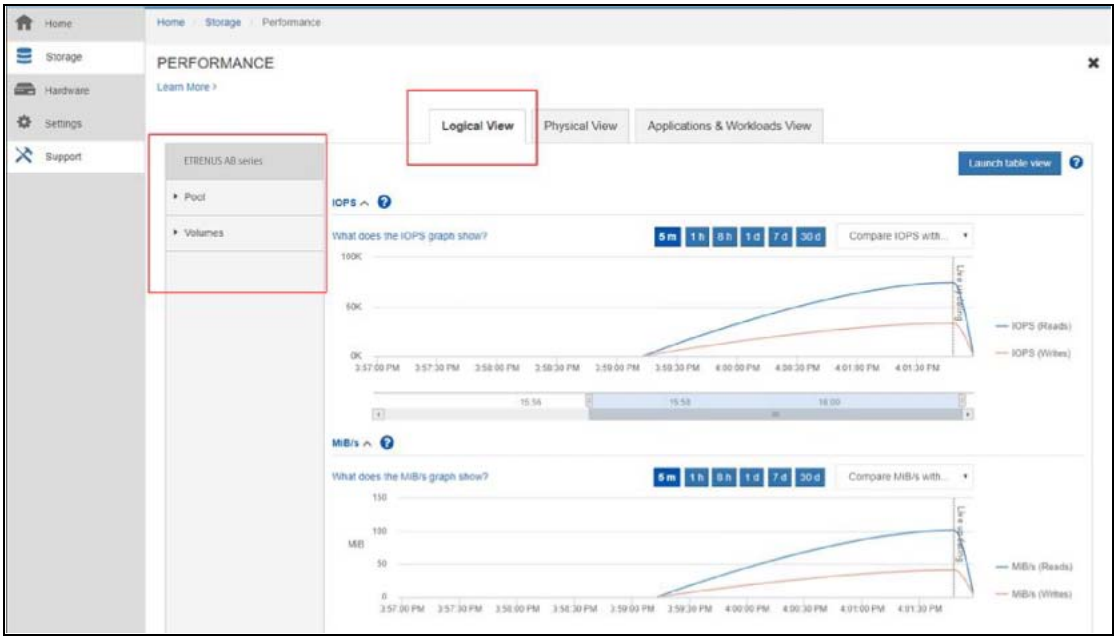
# 7. SANtricity System Manager を使用した ETERNUS AB series のパフォーマンス監視

ストレージシステム操作中は、ストレージシステムのパフォーマンスを監視すると便利です。SANtricity System Manager を使用すると、ETERNUS AB series のパフォーマンスデータをテキスト形式とグラフィカル形式の両方のダッシュボード形式で表示できます。

SANtricity System Manager は、論理レベル、物理レベル、およびアプリケーションレベルのワークロードをキャプチャできる、優れたパフォーマンス監視機能を提供します。アプリケーションとワークロードのタグ付け、パフォーマンスデータの強化、組み込みモニタ、ボリューム使用率のグラフィカル表示などの新機能が追加されています。パフォーマンスモニタを表示するには、SANtricity System Manager で「View Performance Details」を選択します。3 つの主要なカテゴリのパフォーマンスを確認できます。

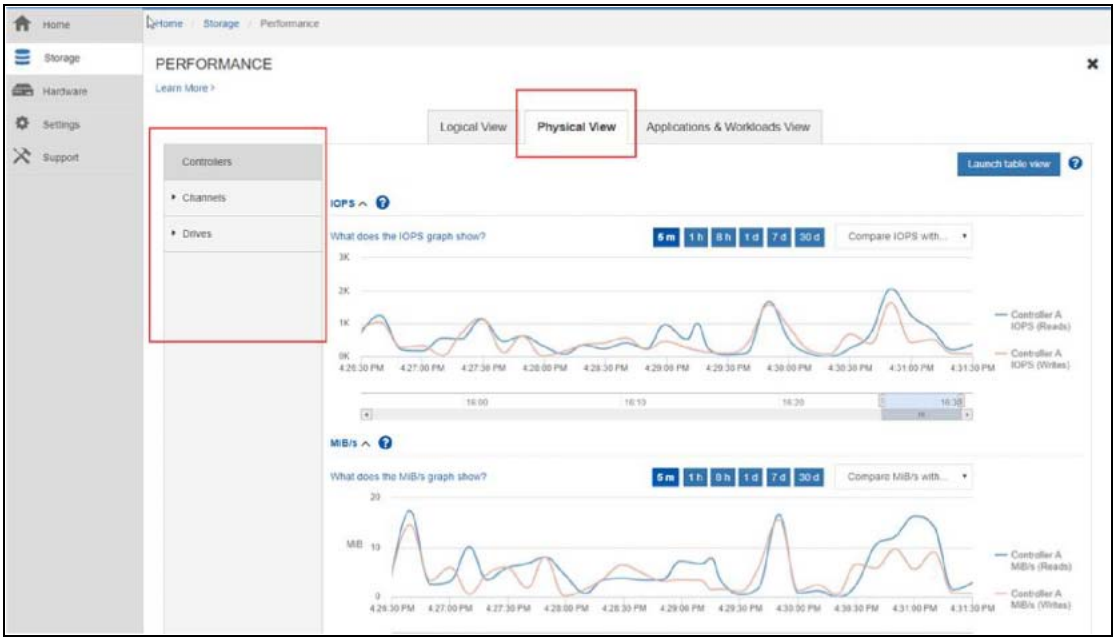
- 「Logical View」を使用すると、プール、ボリュームグループ、またはボリュームによって情報をフィルタリングできます。図 7.1 を参照してください。

図 7.1　Logical View を使用したパフォーマンスモニタの例



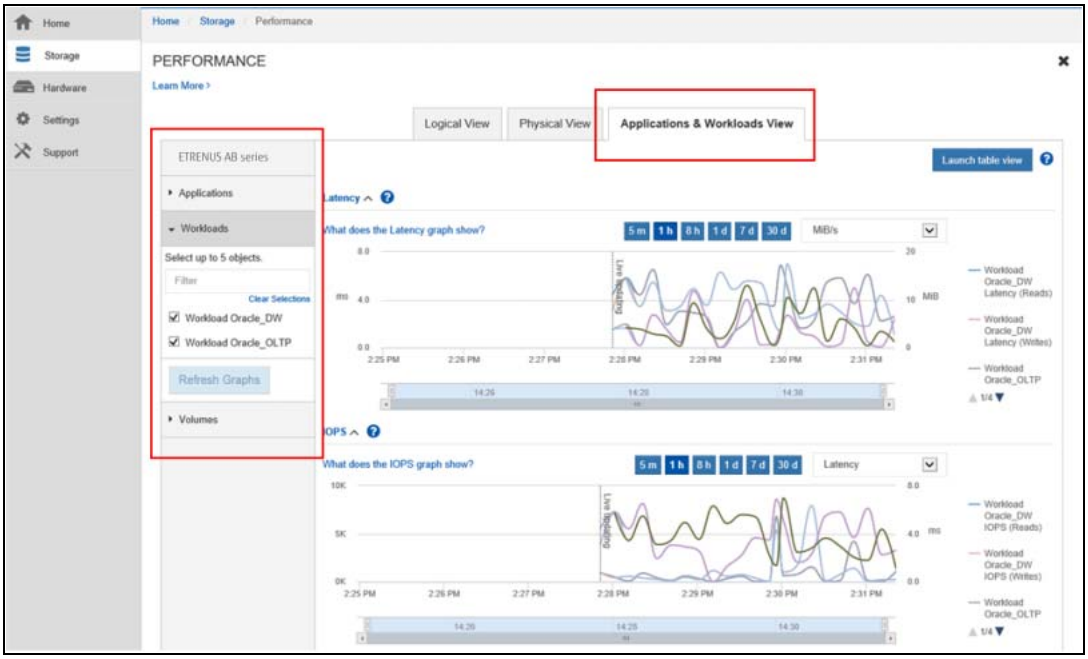
- 「Physical View」を使用すると、ホストチャネルとドライブによって情報をフィルタリングできます。図 7.2 を参照してください。

図 7.2　Physical View を使用したパフォーマンスモニタの例




- 「Applications & Workloads View」を使用すると、アプリケーション、ワークロード、およびワークロード内のボリュームによって情報をフィルタリングできます。図 7.3 を参照してください。

図 7.3　Applications & Workloads View を使用したパフォーマンスモニタの例



# 8. まとめ

本書では、高パフォーマンスの MS SQL Server データベース向けの ETERNUS AB series ソリューションについて説明します。このソリューションでは、クラス最高レベルのエンドツーエンドの最新の SAN および NVMe テクノロジーを活用して、ビジネスクリティカルな IT サービスを今日提供するとともに、将来に備えます。

ETERNUS AB series により、将来のニーズにも対応し、現在も使用可能な SAN アレイを構築しました。また、現在の運用プロセスと手順に容易に実装できます。ETERNUS AB series は、高パフォーマンス、一貫した低レイテンシ、および高度な HA 機能を提供して市場をリードする製品です。

ETERNUS AB series は組み込みの System Manager を使用して容易にプロビジョニングできます。準備するシステムが多数ある場合は、REST API を使用して ETERNUS AB series をプロビジョニングすることもできます。( 付録を参照してください。)

ETERNUS AB series には、堅牢な監視機能も組み込まれています。この機能を使用すると、パフォーマンスの問題を論理レベル、物理レベル、およびアプリケーションレベルでトラブルシューティングできます。

ETERNUS AB series は、非常に厳しいレイテンシ要件を解決するだけでなく、SQL Server データベースに関する課題も次のように解決します。

- 既存のアプリケーションのパフォーマンスを大幅に向上させ、アプリケーションを再設計することなく IOPS あたりのコストを削減します。
- RAID10 および DDP により、SQL Server のパフォーマンスが向上します。
- レスポンスタイムの短縮により、ユーザーの生産性が向上し、ビジネス効率が向上します。
- SQL Server Always On 可用性グループは、高可用性とディザスタリカバリのための代替ソリューションを提供します。

# A. REST API を使用した ETERNUS AB series システムのプロビジョニング

```
# EF-Series System Manager RESTAPI Volume Provisioning

# =========================
# Globals
# =========================

# bypass untrusted certificates -- only do this for sources you trust!
add-type @"
    using System.Net;
    using System.Security.Cryptography.X509Certificates;
    public class TrustAllCertsPolicy : ICertificatePolicy {
        public bool CheckValidationResult(
            ServicePoint srvPoint, X509Certificate certificate,
            WebRequest request, int certificateProblem)
            { return true;
        }
    }
"@
[System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy
[Net.ServicePointManager]::SecurityProtocol = "Tls12, Tls11, Tls, Ssl3"

# IPs and credentials
$EFSeriesC2A = "https://10.251.229.52:8443/devmgr/"
$username = "admin"
$pass = "Infinit1"

# persist web session after logging in
$session = New-Object Microsoft.Powershell.Commands.WebRequestSession

# headers
$headers = @{}
$headers.Add('accept','application/json')
$headers.Add('Content-Type','application/json')

# misc
$storageSystemId = "1"
```

```
# =========================
# Initiate Session
# =========================
# as alternative, can also store creds as plaintext in url:
# Invoke-RestMethod -Method Get -Uri
($EFSeriesC2A+"devmgr/utils/login?uid=admin&pwd=myPassword123&xsrf=false&onlycheck=false") -
Headers $headers

Function LoginAPI ($username, $pass) {
    $credentialsJSON = '
    {
      "userId": "' + $username + '",
      "password": "' + $pass + '"
    }'

    $restParams = @{
        Method     = 'Post'
        Uri        = $EFSeriesC2A+"utils/login"
        Headers    = $headers
        Body       = $credentialsJSON
        WebSession = $session
    }

    Invoke-RestMethod @restParams
}
# LoginAPI $username $pass
```

```
# ========================
# GET Functions
# ========================
# Ex endpoints:
# Invoke-RestMethod -Method Get -Uri ($EFSeriesC2A+"devmgr/utils/about") -Headers $headers
# Invoke-RestMethod -Method Get -Uri ($EFSeriesC2A+"devmgr/utils/buildinfo") -Headers $headers

Function GetStorageSystems {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems"
        Headers    = $headers
        WebSession = $session
    }

    try {
        Invoke-RestMethod @restParams
    }
    catch {
        CatchException $restParams
    }
}
# GetStorageSystems


Function GetEvents {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/events"
        Headers    = $headers
        WebSession = $session
    }

    try {
        Invoke-RestMethod @restParams
    }
    catch {
        CatchException $restParams
    }
}
# GetEvents

Function GetVolumes ($getIdsOnly) {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/volumes"
        Headers    = $headers
        WebSession = $session
    }

    try {
        if ($getIdsOnly -eq $true) {
            (Invoke-RestMethod @restParams).id
        }
        else {
            (Invoke-RestMethod @restParams)
        }
    }
    catch {
        CatchException $restParams
    }
}
# GetVolumes $true # '$false' returns all volume details


Function GetControllerId {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/controllers"
        Headers    = $headers
        WebSession = $session
    }

    try {
        (Invoke-RestMethod @restParams).controllerRef[0]
    }
    catch {
        CatchException $restParams
    }
```

```
        else {
            (Invoke-RestMethod @restParams)
        }
    }
    catch {
        CatchException $restParams
    }
}
# GetDriveIds $true # '$false' returns all drive details

Function GetSnapshots {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/snapshot-images"
        Headers    = $headers
        WebSession = $session
    }

    try {
        (Invoke-RestMethod @restParams)
    }
    catch {
        CatchException $restParams
    }
}
# GetSnapshots}
# GetControllerId

Function GetStoragePoolId {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/storage-pools"
        Headers    = $headers
        WebSession = $session
    }

    try {
        (Invoke-RestMethod @restParams).volumeGroupRef
    }
    catch {
        CatchException $restParams
    }
}
# GetStoragePoolId

Function GetHosts {
    $restParams =
        @{ Method  = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/hosts"
        Headers    = $headers
        WebSession = $session
    }

    try {
        Invoke-RestMethod @restParams
    }
    catch {
        CatchException $restParams
    }
}
# GetHosts

Function GetDriveIds ($getIdsOnly) {
    $restParams = @{
        Method     = 'Get'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/drives"
        Headers    = $headers
        WebSession = $session
    }

    try {
        if ($getIdsOnly -eq $true) {
            (Invoke-RestMethod @restParams).id
        }
```

```
# ========================
# POST Functions
# ========================

Function CreateVolume ($volName, $size) {
    $controllerId = GetControllerId
    $poolId = GetStoragePoolId
    $volumeDetails = '
    {
       "poolId": "' + $poolId + '",
       "name": "' + $volName + '",
       "sizeUnit": "gb",
       "size": "' + $size + '",
       "segSize": 0,
       "dataAssuranceEnabled": false,
       "owningControllerId": "' + $controllerId + '",
       "metaTags": [
        {
            "key": "string",
            "value": "string"
        }
       ]
    }'

    $restParams = @{
        Method     = 'Post'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/volumes"
        Headers    = $headers
        Body       = $volumeDetails
        WebSession = $session
    }

    try {
        Invoke-RestMethod @restParams
    }
  catch {
        CatchException $restParams
    }
}
# CreateVolume delThisVolume 25

Function MapVolume ($targetHostId, $targetVolId) {
    $mappingDetails = '
    {
        "mappableObjectId": "' + $targetVolId + '",
        "targetId": "' + $targetHostId + '"
    }'

    $restParams = @{
        Method     = 'Post'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/volume-mappings"
        Headers    = $headers
        Body       = $mappingDetails
        WebSession = $session
    }

    try {
        Invoke-RestMethod @restParams
    }
    catch {
        CatchException $restParams
    }
}
# MapVolume "84000000600A098000BF6F310030060D5C3C58CA" "02000000600A098000BF85F300001E555C41E60F"
```

```
# =========================
# DELETE Functions
# =========================

Function DeleteVolume ($targetVolId){
    $restParams = @{
        Method     = 'Delete'
        Uri        = $EFSeriesC2A+"v2/storage-systems/"+$storageSystemId+"/volumes/"+$targetVolId
        Headers    = $headers
        Body       = $mappingDetails
        WebSession = $session
    }

    try {
        Invoke-RestMethod @restParams
    }
    catch {
        CatchException $restParams
    }
}
# DeleteVolume 02000000600A098000BF85F300001E615C45CC56



# =========================
# Exception Handler
# =========================

Function CatchException ($restParams) {
    if ($_.Exception.Message -like "*401*") {
        Write-Host "Not logged in -- attempting log in now . . ."
        LoginAPI $username $pass
        Invoke-RestMethod @restParams
    }
    else {
        Write-Host ($_.Exception.Message)
    }
}
```

```
# =========================
# Function Calls
# =========================

# new session:
$session = New-Object Microsoft.Powershell.Commands.WebRequestSession

GetHosts
```

ETERNUS AB series オールフラッシュアレイ
ETERNUS AB series を使用した Microsoft SQL Server のベストプラクティスガイド

P3AG-6612-02Z0

発行年月　2025 年 3 月
発行責任　エフサステクノロジーズ株式会社

- 本書の内容は、改善のため事前連絡なしに変更することがあります。
- 本書の内容は、細心の注意を払って制作致しましたが、本書中の誤字、情報の抜け、本書情報の使用に起因する運用結果に関しましては、責任を負いかねますので予めご了承願います。
- 本書に記載されたデータの使用に起因する第三者の特許権およびその他の権利の侵害については、当社はその責を負いません。
- 無断転載を禁じます。

Fsas Technologies