解決策 の設計を確認します

BeeGFS on NetApp with E-Series Storage

NetApp
January 27, 2026

目次

解決策 の設計を確認します

設計の概要

BeeGFS並列ファイルシステムとNetApp EF600ストレージシステムを組み合わせたNetApp解決策 上でBeeGFSをサポートするには、特定の機器、ケーブル配線、構成が必要です。

詳細はこちら。

- "ハードウェア構成"
- "ソフトウェア構成"
- "設計の検証"
- "サイジングガイドライン"
- "パフォーマンスの調整"

設計とパフォーマンスの違いを伴う派生アーキテクチャ：

- "大容量ビルディングブロック"

ハードウェア構成

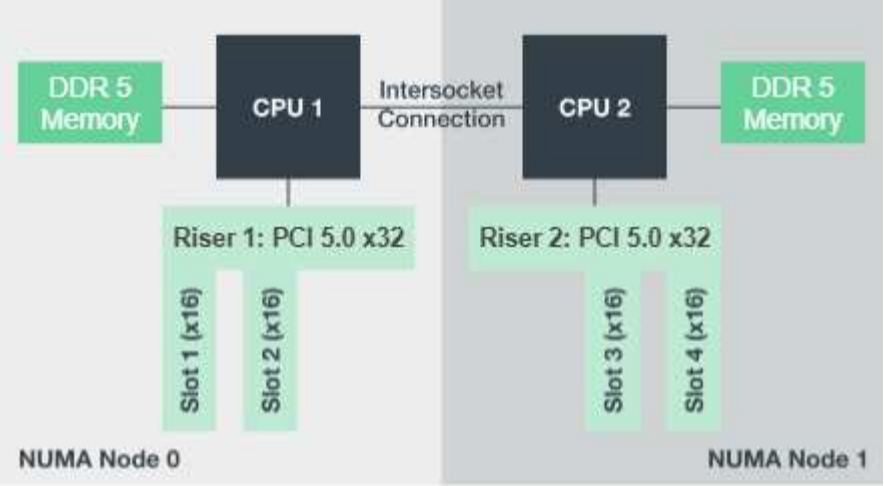
ネットアップのBeeGFSのハードウェア構成には、ファイルノードとネットワークのケーブル配線が含まれます。

ファイルのノード構成

ファイルノードには、別々のNUMAゾーンとして構成された2つのCPUソケットがあり、同じ数のPCIeスロットとメモリへのローカルアクセスが含まれます。

InfiniBandアダプタは、適切なPCIライザーまたはスロットに装着する必要があります。これにより、使用可能なPCIeレーンとメモリチャネル間でワークロードが分散されます。個々のBeeGFSサービスの作業を特定のNUMAノードに完全に分離することで、ワークロードのバランスを調整します。目標は、各ファイルノードのパフォーマンスを、2つの独立したシングルソケットサーバと同様にすることです。

次の図は、ファイルノードのNUMA構成を示しています。



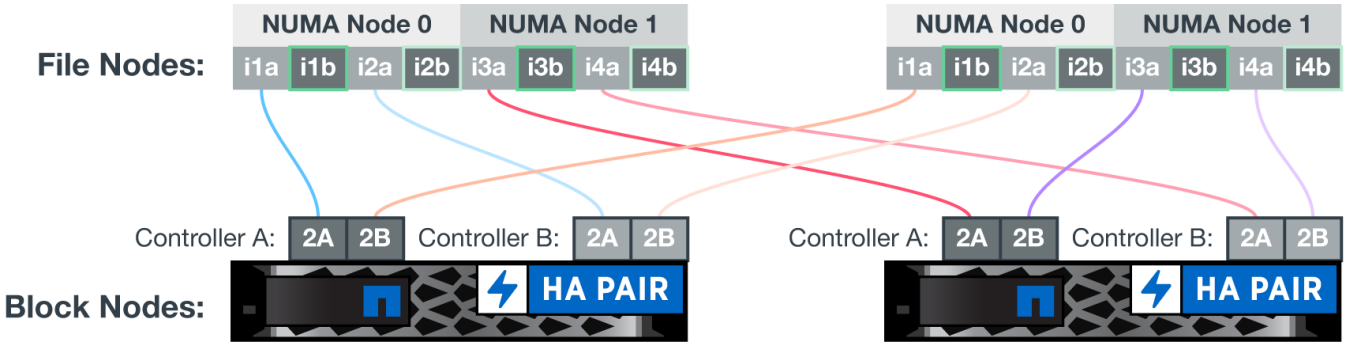
BeeGFSプロセスは、使用するインターフェイスが同じゾーン内にあることを確認するために、特定のNUMAゾーンに固定されます。この構成により、ソケット間接続を介したリモートアクセスが不要になります。ソケット間接続はQPIまたはGMI2リンクと呼ばれることもあります。最新のプロセッサアーキテクチャであっても、HDR InfiniBandなどの高速ネットワークを使用する場合はボトルネックになる可能性があります。

ネットワークのケーブル構成

ビルディングブロック内では、各ファイルノードは合計4つの冗長InfiniBand接続を使用して2つのブロックノードに接続されます。また、各ファイルノードにInfiniBandストレージネットワークへの冗長接続が4つあります。

次の図に注意してください。

- 緑で示されているすべてのファイルノードポートは、ストレージファブリックへの接続に使用されます。他のすべてのファイルノードポートは、ブロックノードへの直接接続です。
- 特定のNUMAゾーン内の2つのInfiniBandポートは、同じブロックノードのAおよびBコントローラに接続します。
- NUMAノード0のポートは常に最初のブロックノードに接続します。
- NUMAノード1のポートは、2番目のブロックノードに接続します。



スプリッタケーブルを使用してストレージスイッチをファイルノードに接続する場合は、1本のケーブルが分岐して薄い緑色のポートに接続する必要があります。もう1本のケーブルが分岐し、濃い緑色で示されているポートに接続します。また、冗長スイッチを使用するストレージネットワークの場合、薄い緑色のポートは1つのスイッチに接続し、濃い緑色のポートは別のスイッチに接続します。

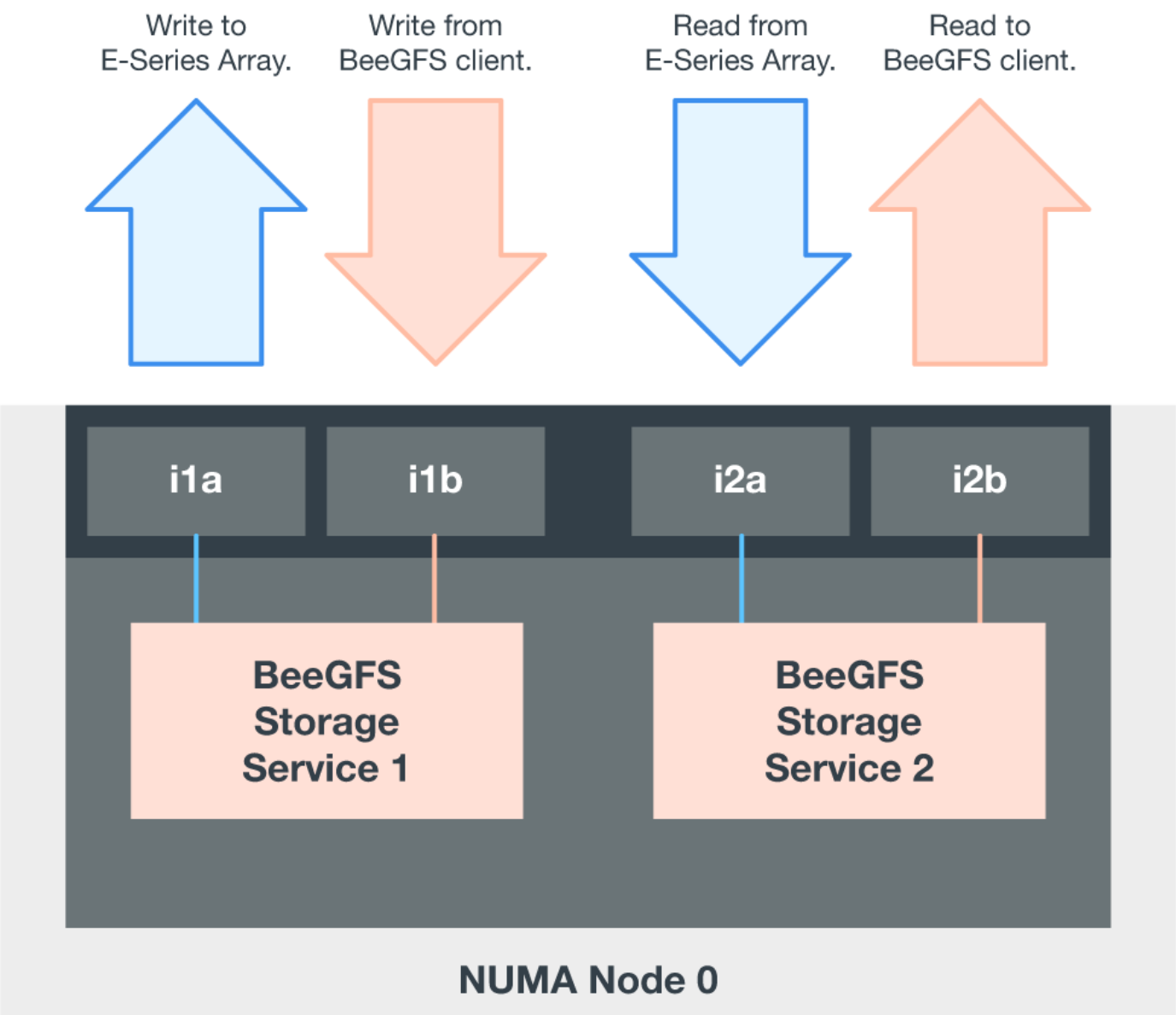
この図に示すケーブル構成では、BeeGFSサービスごとに次のことが可能です。

- BeeGFSサービスを実行しているファイルノードに関係なく、同じNUMAゾーンで実行します。
- 障害が発生した場所に関係なく、フロントエンドストレージネットワークおよびバックエンドブロックノードへのセカンダリの最適パスを確保する。
- ブロックノード内のファイルノードまたはコントローラのメンテナンスが必要な場合は、パフォーマンスへの影響を最小限に抑えます。

帯域幅を活用するためのケーブル接続

PCIeの完全な双方向帯域幅を利用するには、各InfiniBandアダプタの1つのポートをストレージファブリックに接続し、もう1つのポートをブロックノードに接続します。

次の図に、PCIeの双方向帯域幅をフルに活用するためのケーブル配線の設計を示します。



BeeGFSサービスごとに、同じアダプタを使用して、クライアントトラフィックに使用する優先ポートと、そのサービスボリュームのプライマリ所有者であるブロックノードコントローラへのパスを接続します。詳細については、を参照してください "ソフトウェア構成"。

ソフトウェア構成

ネットアップのBeeGFSのソフトウェア設定には、BeeGFSネットワークコンポーネント、EF600ブロックノード、BeeGFSファイルノード、リソースグループ、BeeGFSサービスが含まれます。

BeeGFSネットワーク設定

BeeGFSネットワーク設定は、次のコンポーネントで構成されます。

- *フローティングIP *フローティングIPは、同じネットワーク内の任意のサーバーに動的にルーティングできる一種の仮想IPアドレスです。複数のサーバーが同じフローティングIPアドレスを所有できますが、一

度にアクティブにできるのは1つのサーバーのみです。

各BeeGFSサーバサービスには、BeeGFSサーバサービスの実行場所に応じてファイルノード間を移動できる独自のIPアドレスがあります。このフローティングIP構成では、各サービスが他のファイルノードに独立してフェイルオーバーできます。クライアントは、特定のBeeGFSサービスのIPアドレスを知るだけで済み、そのサービスを現在実行しているファイルノードを認識する必要はありません。

- * BeeGFSサーバのマルチホーミング構成*解決策 の密度を高めるために'各ファイル・ノードには同じIPサブネットにIPが設定された複数のストレージ・インターフェイスがあります

  この設定がLinuxネットワークスタックで正常に機能するようにするには、追加の設定が必要です。これは、デフォルトでは、IPが同じサブネット内にある場合、1つのインターフェイスへの要求に別のインターフェイスで応答できるからです。他の欠点に加えて、このデフォルトの動作により、RDMA接続を適切に確立または維持することができなくなります。

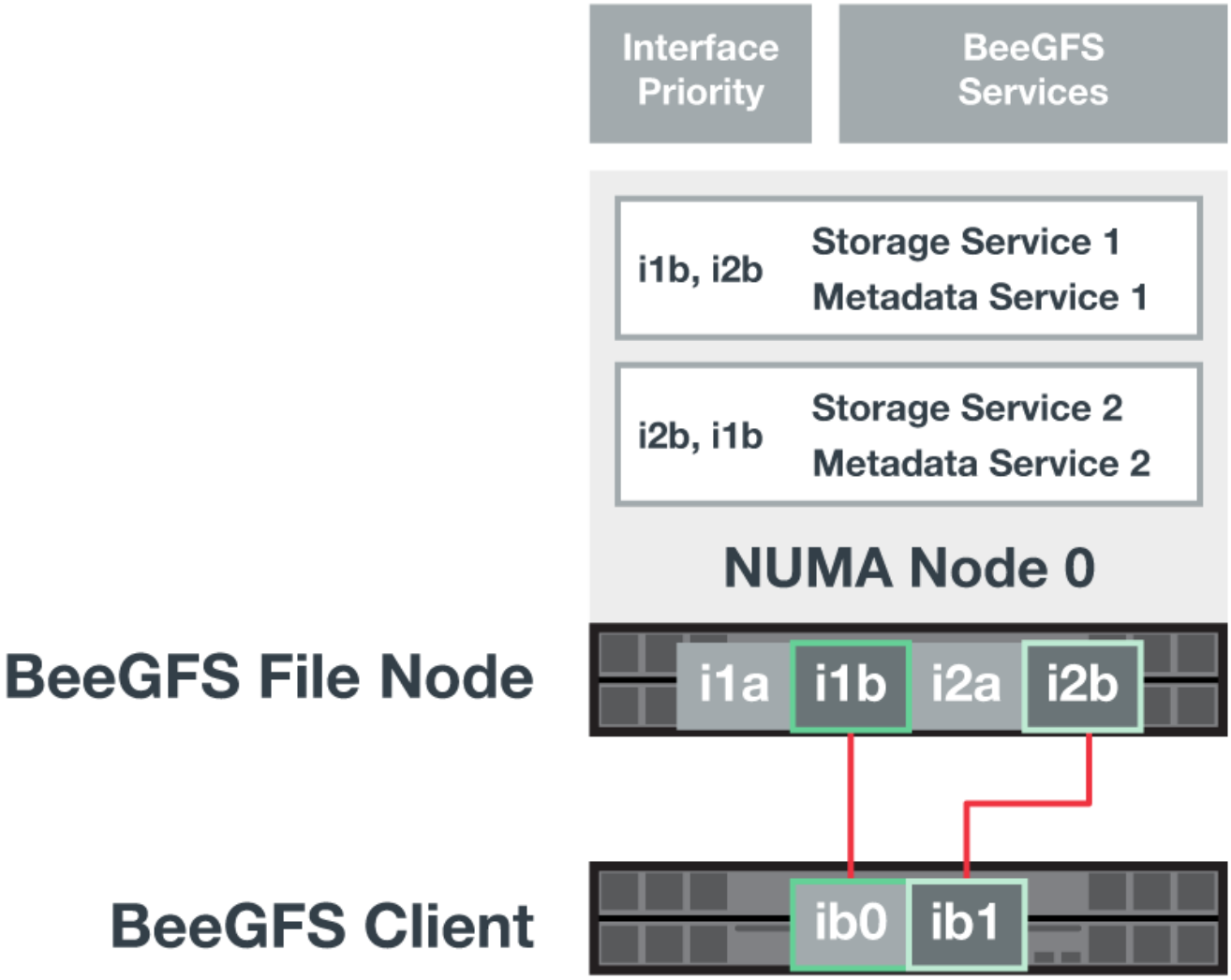
  Ansibleベースの導入では、逆方向パス（RP）とアドレス解決プロトコル（ARP）の動作の締め付けに加え、フローティングIPの開始と停止のタイミングを保証します。対応するIPルートとルールが動的に作成され、マルチホームネットワークの設定が適切に機能できるようになります。

- * BeeGFSクライアントのマルチレール構成*_Multi-rail_とは、アプリケーションが複数の独立したネットワーク接続（「レール」）を使用してパフォーマンスを向上させる機能を指します。

  BeeGFSはマルチレールサポートを実装し、1つのIPoIBサブネットで複数のIBインターフェイスを使用できるようにします。この機能により、RDMA NIC間での動的なロードバランシングなどの機能が有効になり、ネットワークリソースの使用が最適化されます。また、NVIDIA GPUDirect Storage（GDS）とも統合されているため、システム帯域幅が増加し、クライアントのCPUのレイテンシと使用率が低下します。

  このマニュアルでは、単一のIPoIBサブネット設定の手順について説明します。デュアルIPoIBサブネット構成はサポートされていますが、シングルサブネット構成と同じ利点はありません。

  次の図に、複数のBeeGFSクライアントインターフェイス間でのトラフィックの分散を示します。



BeeGFSの各ファイルは通常、複数のストレージサービスにまたがってストライピングされるため、マルチレール構成では、単一のInfiniBandポートでは実現できないスループットよりも高いスループットをクライアントに提供できます。たとえば、次のコード例は、クライアントが両方のインターフェイス間でトラフィックを分散できるようにする、一般的なファイルストライピング設定を示しています。

+

```
root@beegfs01:/mnt/beegfs# beegfs-ctl --getentryinfo myfile
Entry type: file
EntryID: 11D-624759A9-65
Metadata node: meta_01_tgt_0101 [ID: 101]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 1M
+ Number of storage targets: desired: 4; actual: 4
+ Storage targets:
  + 101 @ stor_01_tgt_0101 [ID: 101]
  + 102 @ stor_01_tgt_0101 [ID: 101]
  + 201 @ stor_02_tgt_0201 [ID: 201]
  + 202 @ stor_02_tgt_0201 [ID: 201]
```

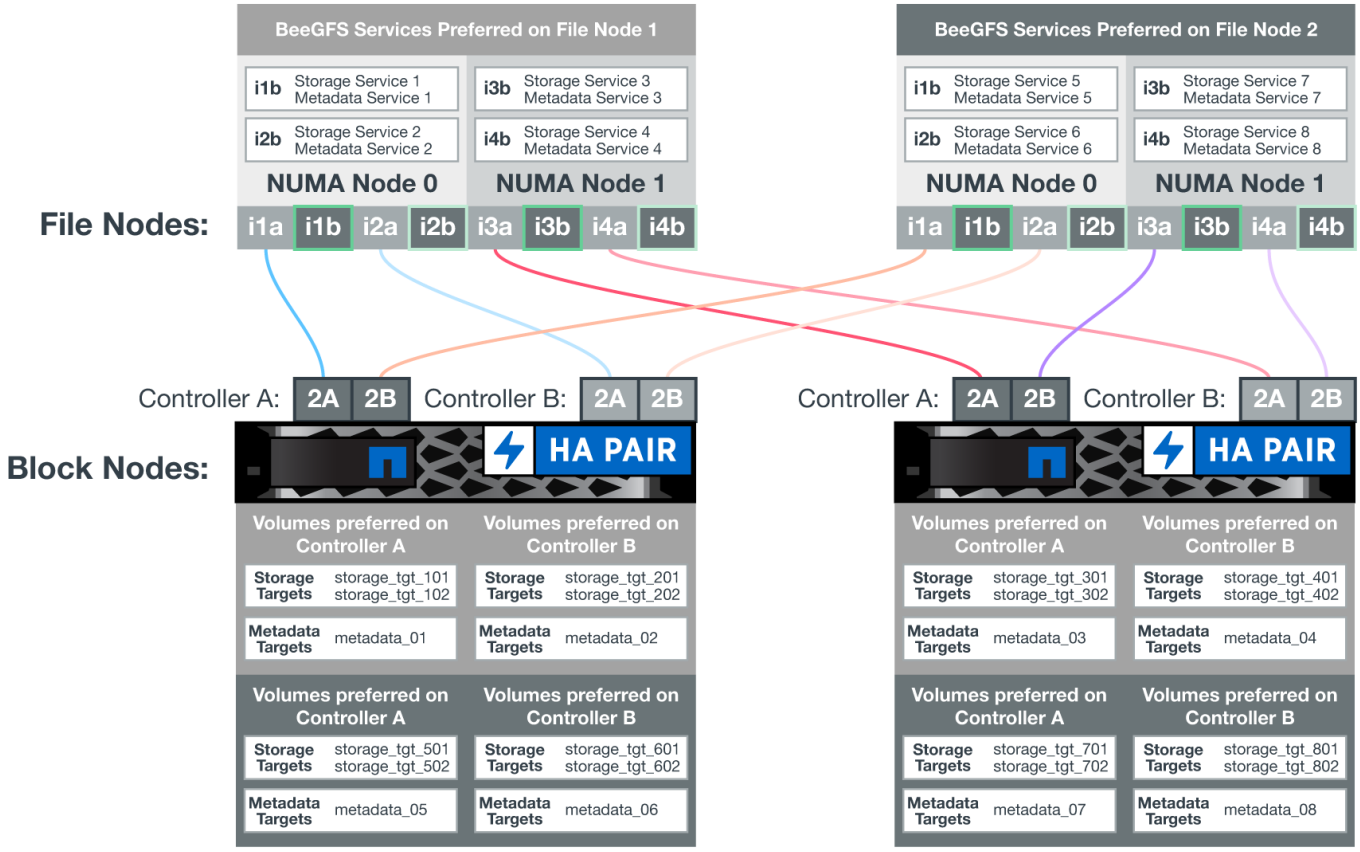
EF600ブロックノード構成

ブロックノードは、同じドライブセットへの共有アクセスを持つ2つのアクティブ/アクティブRAIDコントローラで構成されます。通常、各コントローラにはシステムに設定されているボリュームの半分が所有されますが、必要に応じてもう一方のコントローラがテイクオーバーできます。

ファイルノード上のマルチパスソフトウェアは、各ボリュームへのアクティブなパスと最適パスを決定し、ケーブル、アダプタ、またはコントローラに障害が発生した場合に自動的に代替パスに移動します。

次の図は、EF600ブロックノードのコントローラのレイアウトを示しています。

共有ディスクのHA解決策 を使用する場合、ボリュームが両方のファイルノードにマッピングされ、必要に応じて相互にテイクオーバーできるようになります。次の図は、パフォーマンスを最大化するためにBeeGFSサービスと優先ボリューム所有権を設定する例を示しています。各BeeGFSサービスの左側にあるインターフェイスは、クライアントとその他のサービスが接続に使用する優先インターフェイスを示します。



前の例では、クライアントとサーバサービスは、インターフェイスi1bを使用してストレージサービス1と通信することを好みます。ストレージサービス1は、最初のブロックノードのコントローラA上のボリューム（storage_tgt_101、102）と通信するための優先パスとしてインターフェイスi1aを使用します。この構成では、InfiniBandアダプタで使用できる完全な双方向PCIe帯域幅を使用し、PCIe 4.0では実現できないデュアルポートHDR InfiniBandアダプタよりも優れたパフォーマンスを実現します。

BeeGFSファイルのノード設定

BeeGFSファイルノードは、複数のファイルノード間でBeeGFSサービスのフェイルオーバーを容易にするために、ハイアベイラビリティ（HA）クラスタに構成されます。

HAクラスタは、広く使用されている2つのLinux HAプロジェクトに基づいて設計されています。クラスタメンバーシップ用のCorosyncと、クラスタリソース管理用のPacemakerです。詳細については、を参照してください "高可用性アドオンに関するRed Hatトレーニング"。

複数のオープンクラスタフレームワーク（OCF）リソースエージェントがネットアップにオーサリングされ、拡張されました。このエージェントを使用すると、クラスタでBeeGFSリソースのインテリジェントな起動と監視が可能になります。

BeeGFS HA clusters（BeeGFS HAクラスタ

通常、BeeGFSサービスを（HAの有無にかかわらず）開始するときは、次のようなリソースが必要になります。

- サービスに到達できるIPアドレス。通常はNetwork Managerによって設定されます。
- BeeGFSがデータを格納するためのターゲットとして使用する基盤となるファイルシステム。

これらは通常'/etc/fstabで定義され’システム・ディスクでマウントされます

- 他のリソースの準備ができたときにBeeGFSプロセスを開始するシステムdサービス。

  追加のソフトウェアがない場合、これらのリソースは単一のファイルノードからのみ開始されます。したがって、ファイルノードがオフラインになると、BeeGFSファイルシステムの一部にアクセスできなくなります。

複数のノードでBeeGFSサービスを起動できるため、ペースメーカーは各サービスと依存するリソースが一度に1つのノードでのみ動作していることを確認する必要があります。たとえば、2つのノードが同じBeeGFSサービスを起動しようとすると、どちらも基盤となるターゲット上の同じファイルに書き込みを試みると、データが破損するおそれがあります。この状況を回避するために、Pacemakerは、クラスタ全体の状態をすべてのノードで確実に同期し、クォーラムを確立するために、Corosyncに依存しています。

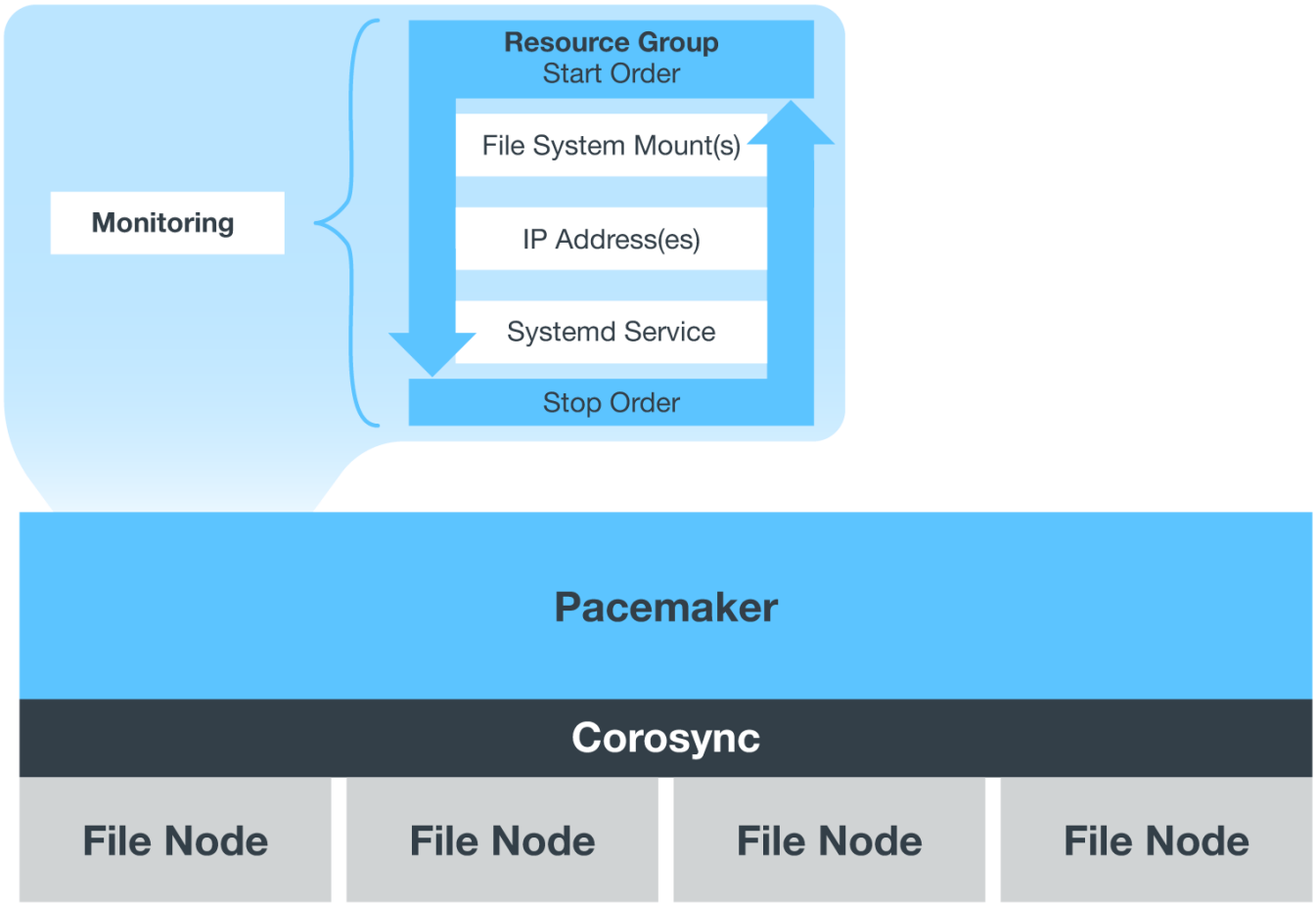
クラスタで障害が発生すると、Pacemakerは別のノードのBeeGFSリソースに反応して再起動します。一部の状況では、ペースメーカーが障害のある元のノードと通信できず、リソースが停止していることを確認できない場合があります。BeeGFSリソースを他の場所から再起動する前にノードが停止していることを確認するために、Pacemakerは障害のあるノードをフェンシングします。この場合、電源を切断します。

多くのオープンソースフェンシングエージェントを使用すると、Pacemakerは配電ユニット（PDU）を搭載したノードを遮断したり、RedfishなどのAPIを搭載したサーバベースボード管理コントローラ（BMC）を使用してノードを遮断したりできます。

HAクラスタでBeeGFSを実行している場合は、すべてのBeeGFSサービスと基盤となるリソースがペースメーカーによってリソースグループで管理されます。各BeeGFSサービスとそれが依存するリソースは、リソースグループに設定されます。これにより、リソースが正しい順序で開始および停止され、同じノードに配置されるようになります。

BeeGFSリソースグループごとに、PacemakerはカスタムのBeeGFSモニタリングリソースを実行します。このリソースは、障害状態を検出し、特定のノードでBeeGFSサービスがアクセスできなくなったときにフェイルオーバーをインテリジェントにトリガーします。

次の図に、Pacemaker制御のBeeGFSサービスと依存関係を示します。



同じタイプの複数のBeeGFSサービスが同じノードで起動するように、Pacemakerはマルチモード設定方式を使用してBeeGFSサービスを開始するように設定されます。詳細については、を参照してください "マルチモードでのBeeGFSのマニュアル"。

BeeGFSサービスは複数のノードで起動できる必要があるため'各サービスの構成ファイル（通常は/etc/beegfsにあります）は'そのサービスのBeeGFSターゲットとして使用されるEシリーズボリュームの1つに保存されますこれにより、特定のBeeGFSサービスのデータとともに、サービスの実行に必要なすべてのノードから設定へのアクセスが可能になります。

```
# tree stor_01_tgt_0101/ -L 2
stor_01_tgt_0101/
├── data
│   ├── benchmark
│   ├── buddymir
│   ├── chunks
│   ├── format.conf
│   ├── lock.pid
│   ├── nodeID
│   ├── nodeNumID
│   ├── originalNodeID
│   ├── targetID
│   └── targetNumID
└── storage_config
    ├── beegfs-storage.conf
    ├── connInterfacesFile.conf
    └── connNetFilterFile.conf
```

設計の検証

NetApp解決策 のBeeGFSの第2世代設計は、3つのビルディングブロック構成プロファイルを使用して検証されました。

構成プロファイルには、次のものが含まれます。

- BeeGFSの管理、メタデータ、ストレージサービスなど、単一のベースとなるビルディングブロックです。
- BeeGFSメタデータとストレージビルディングブロック
- BeeGFSストレージ専用のビルディングブロック。

ビルディングブロックは、2台のNVIDIA Quantum InfiniBand（MQM8700）スイッチに接続されています。また、10個のBeeGFSクライアントもInfiniBandスイッチに接続し、総合ベンチマークユーティリティの実行に使用しました。

次の図は、NetApp解決策 でBeeGFSを検証するために使用するBeeGFS設定を示しています。

BeeGFSファイルのストライピング

並列ファイルシステムの利点は、複数のストレージターゲットに個別のファイルをストライプすることです。これは、同一または異なる基盤となるストレージシステム上のボリュームを表すことができます。

BeeGFSでは'各ファイルに使用するターゲットの数を制御し'各ファイルストライプに使用するチャンクサイズ（またはブロックサイズ）を制御するために'ディレクトリ単位およびファイル単位でストライピングを構成できますこの構成では、サービスを再設定したり再開したりすることなく、ファイルシステムがさまざまなタイプのワークロードやI/Oプロファイルをサポートできます。ストライプ設定を適用するには'beegfs-ctl'コマンドラインツールを使用するか'ストライピングAPIを使用するアプリケーションを使用します詳細については、のBeeGFSのマニュアルを参照してください "ストライピング" および "ストライピングAPI"。

最高のパフォーマンスを得るために、テスト全体を通してストライプパターンを調整し、各テストに使用するパラメータを記録しました。

IOR帯域幅テスト：複数のクライアント

IOR帯域幅テストでは、OpenMPIを使用して、統合I/OジェネレータツールIOR（から入手可能）の並列ジョブを実行しました "HPC GitHub"）を使用して、10のすべてのクライアントノードから1つ以上のBeeGFSビルディングブロックにアクセスします。特に明記されていない限り：

- すべてのテストで、直接I/Oを使用し、転送サイズは1MiBに設定しました。
- BeeGFSファイルのストライピングは'1 MBのチャンクサイズと1ファイルあたり1つのターゲットに設定されました

IORでは次のパラメータを使用し、1つのビルディングブロックではアグリゲートのファイルサイズが5TiB、3つのビルディングブロックでは40TiBにセグメント数を調整しました。

```
mpirun --allow-run-as-root --mca btl tcp -np 48 -map-by node -hostfile
10xnodes ior -b 1024k --posix.odirect -e -t 1024k -s 54613 -z -C -F -E -k
```

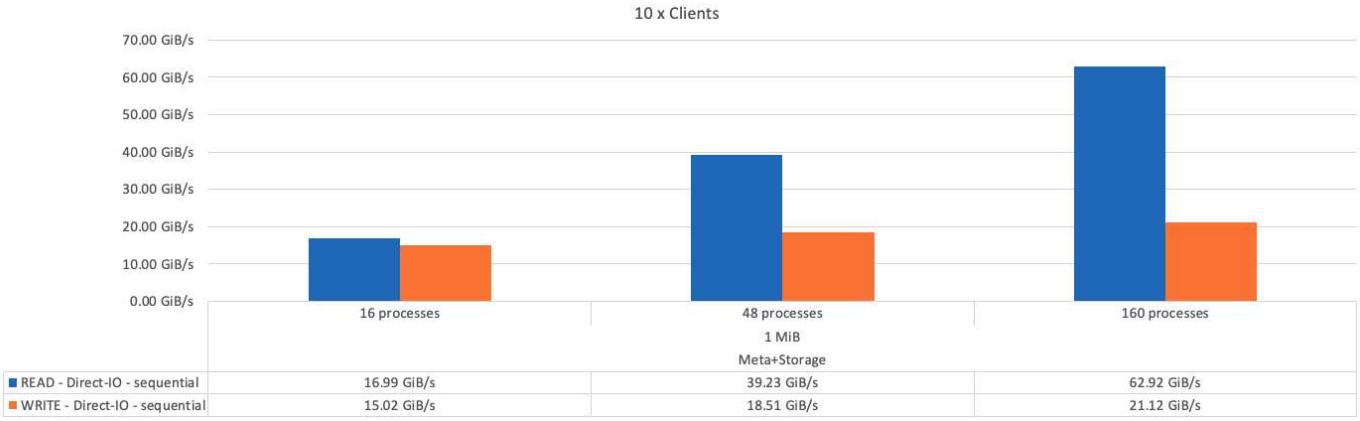
BeeGFSベース（管理、メタデータ、ストレージ）ビルディングブロック**×1**

次の図に、1つのBeeGFSベース（管理、メタデータ、ストレージ）ビルディングブロックを使用したIORテ

スト結果を示します。

BeeGFSメタデータとストレージビルディングブロック

次の図は、1つのBeeGFSメタデータとストレージビルディングブロックを使用したIORテスト結果を示しています。

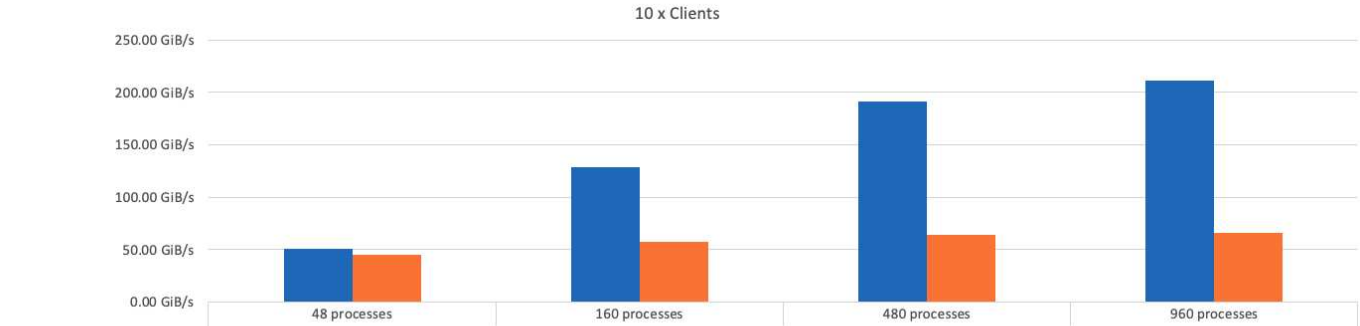


BeeGFSストレージ専用のビルディングブロック

次の図に、1つのBeeGFSストレージ専用ビルディングブロックを使用したIORテスト結果を示します。

3つの**BeeGFS**ビルディングブロック

次の図に、3つのBeeGFSビルディングブロックを使用したIORテスト結果を示します。



基本となるビルディングブロックと後続のメタデータとストレージビルディングブロックのパフォーマンスの違いは、想定どおりごくわずかです。メタデータとストレージのビルディングブロックおよびストレージ専用のビルディングブロックを比較した場合、ストレージターゲットとして使用される追加のドライブが原因で読み取りパフォーマンスがわずかに向上します。ただし、書き込みパフォーマンスに大きな違いはありません。パフォーマンスを向上させるには、複数のビルディングブロックを1つに追加して、パフォーマンスを直線的に拡張します。

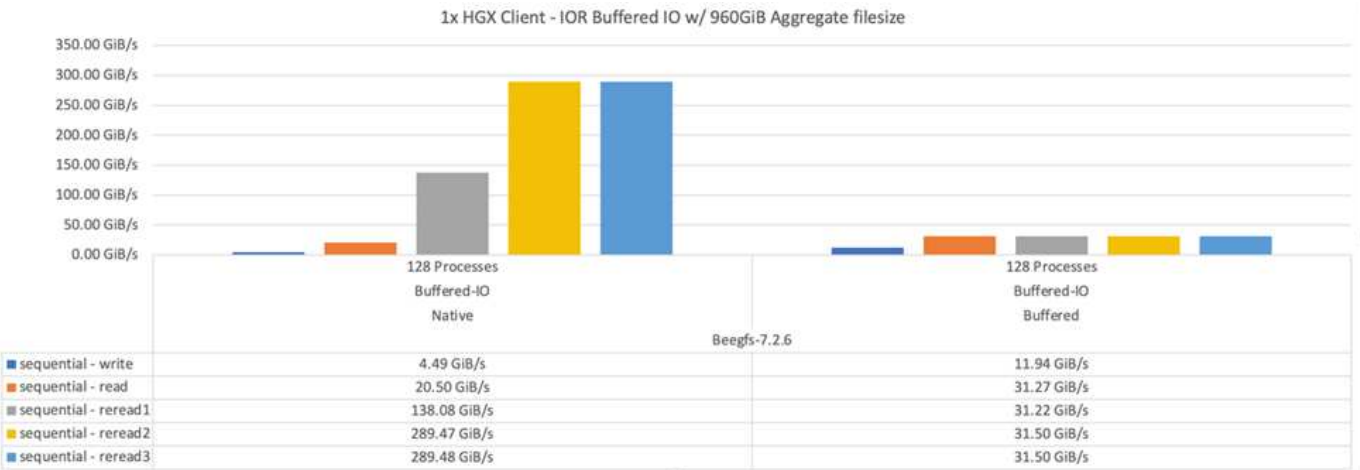
IOR帯域幅テスト：単一クライアント

IOR帯域幅テストでは、OpenMPIを使用して、1台の高性能GPUサーバを使用して複数のIORプロセスを実行し、1台のクライアントで実現可能なパフォーマンスを検証しました。

このテストでは'クライアントがLinuxカーネルのページキャッシュ(`tuneFileCacheType=native')を使用するように構成されている場合に'BeeGFSの再読み取り動作とパフォーマンスを'デフォルトのバッファ設定と比較します

ネイティブ・キャッシュ・モードでは'クライアント上のLinuxカーネル・ページ・キャッシュを使用するため'ネットワーク上で再送信されるのではなく'ローカル・メモリから再読み取り操作を行うことができます

次の図は、3つのBeeGFSビルディングブロックと1つのクライアントを使用したIORテスト結果を示しています。



これらのテストのBeeGFSストライピングは'ファイルあたり8つのターゲットを持つ1MBのチャンクサイズに設定されました

デフォルトのバッファモードを使用した場合の書き込みと初期の読み取りパフォーマンスは向上しますが、同じデータを複数回再読み取りするワークロードでは、ネイティブのキャッシュモードを使用するとパフォーマンスが大幅に向上します。ディープラーニングなどのワークロードで多くの期間にわたって同じデータセットを複数回再読み取りする場合は、読み取りパフォーマンスの向上が重要になります。

メタデータパフォーマンステスト

メタデータパフォーマンステストでは、MDTestツール（IORの一部として含まれる）を使用してBeeGFSのメタデータパフォーマンスを測定しました。テストでは、OpenMPIを使用して、10個のクライアントノードすべてで並列ジョブを実行しました。

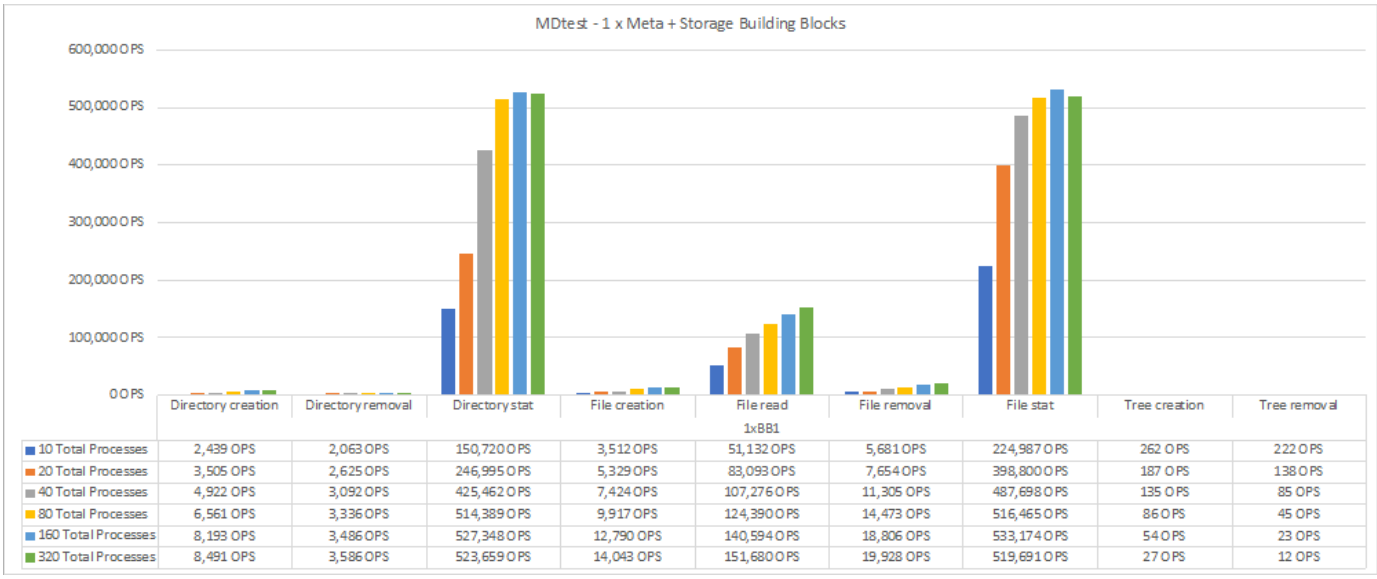
以下のパラメータを使用して、ステップ2xでステップ10から320まで、ファイルサイズ4Kのプロセスの合計数を使用して、ベンチマークテストを実行しました。

```
mpirun -h 10xnodes –map-by node np $processes mdtest -e 4k -w 4k -i 3 -I
16 -z 3 -b 8 -u
```

メタデータパフォーマンスは、まずメタデータとストレージのビルディングブロックを2つずつ使用して測定し、ビルディングブロックを追加してパフォーマンスがどのようにスケールアップするかを示しました。

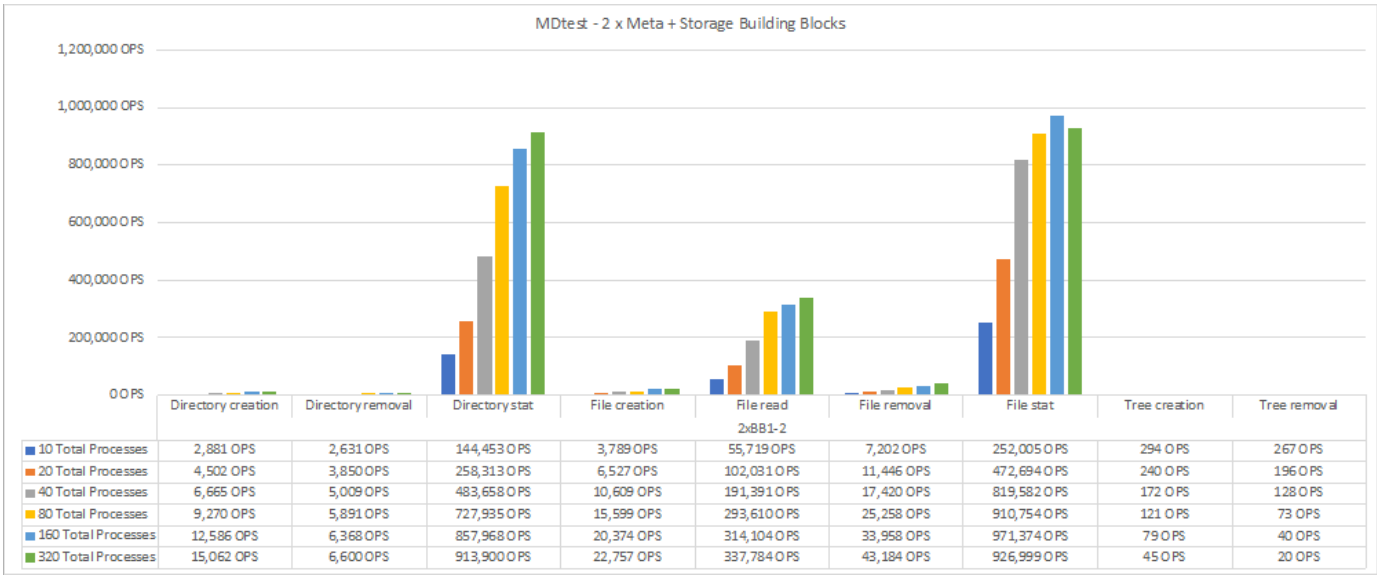
BeeGFSメタデータとストレージビルディングブロック**×1**

次の図は、BeeGFSメタデータとストレージビルディングブロックを1つずつ使用したMDTestの結果を示しています。



BeeGFSメタデータとストレージビルディングブロックが**2**つずつ搭載されています

次の図は、2つのBeeGFSメタデータとストレージビルディングブロックを使用したMDTestの結果を示しています。



	Directory creation	Directory removal	Directory stat	File creation	File read	File removal	File stat	Tree creation	Tree removal
					2xBB1-2				
10 Total Processes	2,881 OPS	2,631 OPS	144,453 OPS	3,789 OPS	55,719 OPS	7,202 OPS	252,005 OPS	294 OPS	267 OPS
20 Total Processes	4,502 OPS	3,850 OPS	258,313 OPS	6,527 OPS	102,031 OPS	11,446 OPS	472,694 OPS	240 OPS	196 OPS
40 Total Processes	6,665 OPS	5,009 OPS	483,658 OPS	10,609 OPS	191,391 OPS	17,420 OPS	819,582 OPS	172 OPS	128 OPS
80 Total Processes	9,270 OPS	5,891 OPS	727,935 OPS	15,599 OPS	293,610 OPS	25,258 OPS	910,754 OPS	121 OPS	73 OPS
160 Total Processes	12,586 OPS	6,368 OPS	857,968 OPS	20,374 OPS	314,104 OPS	33,958 OPS	971,374 OPS	79 OPS	40 OPS
320 Total Processes	15,062 OPS	6,600 OPS	913,900 OPS	22,757 OPS	337,784 OPS	43,184 OPS	926,999 OPS	45 OPS	20 OPS

機能検証

このアーキテクチャの検証の一環として、ネットアップは次の機能テストをいくつか実施しました。

- スイッチポートを無効にして、単一のクライアントInfiniBandポートを障害状態にします。
- スイッチポートを無効にして、単一サーバのInfiniBandポートを障害状態にします。
- BMCを使用した即時サーバ電源オフのトリガー
- ノードを正常にスタンバイにし、別のノードにサービスをフェイルオーバーします。
- ノードを正常にオンラインに戻し、元のノードにサービスをフェイルバックします。
- PDUを使用している一方のInfiniBandスイッチの電源をオフにします。すべてのテストは、BeeGFSクライアントで設定された「sysSessionChecksEnabled：false」パラメータを使用して、ストレステストの実行中に実行されました。エラーやI/Oの中断は発生しませんでした。

既知の問題 がある（を参照） "変更ログ") BeeGFSクライアント/サーバRDMA接続が予期せず中断される場合は、プライマリインターフェイスの喪失（「connInterfacesFile」で定義）またはBeeGFSサーバの障害のいずれかによって、アクティブなクライアントI/Oが最大10分間ハングアップしてから再開します。この問題 は、計画的メンテナンスのためにBeeGFSノードが正常に配置され、スタンバイ状態から外れたとき、またはTCPが使用中のときは発生しません。

NVIDIA DGX SuperPODとBasePODの検証

ネットアップでは、3つのビルディングブロックにメタデータとストレージ構成プロファイルが適用されたBeeGFSファイルシステムを使用して、NVDIA DGX A100 SuperPOD向けのストレージ解決策 の検証を実施しました。認定には、このNVAで説明した解決策 を、さまざまなストレージ、機械学習、ディープラーニングのベンチマークを実行している20台のDGX A100 GPUサーバでテストすることが含まれます。NVIDIAのDGX A100 SuperPODで確立された検証に基づいて、BeeGFS on NetAppソリューションは、DGX SuperPOD H100、H200、B200システムでの使用が承認されました。この拡張は、NVIDIA DGX A100で検証された、以前に確立されたベンチマークとシステム要件を満たすことに基づいています。

詳細については、を参照してください "NVIDIA DGX SuperPODとネットアップ" および "NVIDIA DGX BasePOD"。

サイジングガイドライン

BeeGFS解決策 には、検証テストに基づくパフォーマンスと容量のサイジングに関する推奨事項が含まれます。

ビルディングブロックアーキテクチャの目的は、特定のBeeGFSシステムの要件を満たす複数のビルディングブロックを追加することで、サイズを簡単に決定できる解決策 を作成することです。以下のガイドラインを使用して、環境の要件を満たすために必要なBeeGFSビルディングブロックの数とタイプを見積もります。

これらの見積もりは、ケースのパフォーマンスが最も優れていることに留意してください。統合ベンチマークアプリケーションは、基盤となるファイルシステムの使用を最適化するために作成され、実際のアプリケーションでは使用されない可能性があります。

パフォーマンスのサイジング

次の表に、推奨されるパフォーマンスサイジングを示します。

構成プロファイル	読み取り：**1MiB**	**1MiB**の書き込み
メタデータ+ストレージ	62GiBps	21GiBps
ストレージのみ	64GiBps	21GiBps

メタデータ容量のサイジングの推定値は、500GBの容量でBeeGFSに約1億5、000万ファイルを格納できる「経験則」に基づいています。（詳細については、のBeeGFSのマニュアルを参照してください "システム要件". ）

アクセス制御リストやディレクトリあたりのディレクトリ数やファイル数などの機能を使用すると、メタデータスペースの消費速度にも影響します。ストレージ容量の概算値は、使用可能なドライブ容量と、RAID 6およびXFSオーバーヘッドを考慮しています。

メタデータおよびストレージのビルディングブロックの容量サイジング

次の表に、メタデータ用の推奨容量およびストレージのビルディングブロックを示します。

ドライブサイズ（**2+2 RAID 1**）のメタデータボリュームグループ	メタデータ容量（ファイル数）	ドライブサイズ（**8+2 RAID 6**）ストレージボリュームグループ	ストレージ容量（ファイル内容）
1.92TB	1,938,577,200	1.92TB	51.77TB
3.84TB	3,880,388,400	3.84TB	103.55TB
7.68TB	8、125、278,000	7.68TB	216.74TB
15.3TB	17、269、854、000	15.3TB	460.60TB

メタデータとストレージビルディングブロックのサイジングを行う場合、メタデータボリュームグループとストレージボリュームグループに使用するドライブを小型化することでコストを削減できます。

ストレージ専用のビルディングブロックの容量サイジング

次の表に、ストレージ専用のビルディングブロックの容量をルールベースの設定値で示します。

ドライブサイズ（**10+2 RAID 6**）のストレージボリュームグループ	ストレージ容量（ファイル内容）
1.92TB	59.89TB
3.84TB	119.80TB
7.68TB	251.89TB
15.3TB	538.55TB

グローバルファイルロックが有効になっていないかぎり、ベース（1番目）のビルディングブロックに管理サービスを含める場合のパフォーマンスと容量のオーバーヘッドは最小限です。

パフォーマンスの調整

BeeGFS解決策 には、検証テストに基づいたパフォーマンス調整の推奨事項が含まれます。

BeeGFSは設定不要で妥当なパフォーマンスを提供しますが、パフォーマンスを最大化するために推奨されるチューニングパラメータを一連開発しました。これらのパラメータには、基盤となるEシリーズのブロックノードの機能と、共有ディスクのHAアーキテクチャでBeeGFSを実行するために必要な特別な要件が考慮されています。

ファイルノードのパフォーマンス調整

設定可能なチューニングパラメータには、次のものがあります。

1. *ファイルノードのUEFI/BIOSのシステム設定。*パフォーマンスを最大化するには、ファイルノードとして使用するサーバーモデルのシステム設定を構成することをお勧めします。システム設定は、ベースボード管理コントローラ（BMC）が提供するセットアップユーティリティ（UEFI / BIOS）またはRedfish APIを使用してファイルノードをセットアップするときに設定します。

   システム設定は、ファイルノードとして使用するサーバーモデルによって異なります。使用中のサーバモデルに基づいて、設定を手動で行う必要があります。検証済みのLenovo SR665 V3ファイルノードのシステム設定を構成する方法については、以下を参照してください。 "パフォーマンスのファイルノードシステム設定を調整します" 。

2. *必須構成パラメータのデフォルト設定。*必要な構成パラメータは、BeeGFSサービスの構成方法、およびペースメーカーによるEシリーズボリューム（ブロックデバイス）のフォーマットとマウント方法に影響します。これらの必須設定パラメータには、次のものがあります。

   - BeeGFSサービスの設定パラメータ

     必要に応じて、設定パラメータのデフォルト設定を上書きできます。特定のワークロードやユースケースに合わせて調整できるパラメータについては、を参照して "BeeGFSサービスの設定パラメータ" ください。

   - ボリュームのフォーマットとマウントのパラメータは推奨されるデフォルトに設定されており、高度

なユースケースでのみ調整する必要があります。デフォルト値では、次の処理が実行されます。

- ターゲットタイプ（管理、メタデータ、ストレージなど）、基盤となるボリュームのRAID構成とセグメントサイズに基づいて、ボリュームの初期フォーマットを最適化します。
- Pacemakerが各ボリュームをマウントする方法を調整し、変更がEシリーズのブロックノードにすぐにフラッシュされるようにします。これにより、アクティブな書き込みが進行中の状態でファイルノードに障害が発生してもデータが失われることはありませ

  特定のワークロードやユースケースに合わせて調整できるパラメータについては、を参照して "ボリュームのフォーマットと構成パラメータのマウント"ください。

3. *ファイルノードにインストールされているLinux OSのシステム設定。*の手順4でAnsibleインベントリを作成するときに、Linux OSのデフォルトのシステム設定を上書きできます "Ansibleインベントリを作成します"。

   デフォルトの設定を使用してNetApp解決策 のBeeGFSが検証されましたが、特定のワークロードやユースケースに合わせて調整するように変更することができます。変更可能なLinux OSのシステム設定には、次のようなものがあります。

   - Eシリーズのブロックデバイスの I/Oキュー。

     BeeGFSターゲットとして使用されるEシリーズのブロックデバイスでは、次の目的でI/Oキューを設定できます。

     - デバイスタイプ（NVMe、HDDなど）に基づいてスケジューリングアルゴリズムを調整します。
     - 未処理の要求数を増やします。
     - 要求サイズを調整します。
     - 先読み動作を最適化します。
   - 仮想メモリの設定。

     仮想メモリの設定を調整して、最適な持続ストリーミングパフォーマンスを得ることができます。

   - CPU設定。

     CPU周波数ガバナおよびその他のCPU構成を調整して、パフォーマンスを最大限に高めることができます。

   - 読み取り要求のサイズ

     NVIDIA HCAの最大読み取り要求サイズを増やすことができます。

ブロックノードのパフォーマンス調整

特定のBeeGFSビルディングブロックに適用される構成プロファイルに基づいて、ブロックノードに設定されたボリュームグループが少し変化します。たとえば、24ドライブのEF600ブロックノードがある場合、次のようになります。

- BeeGFSの管理、メタデータ、ストレージサービスなど、単一のベースビルディングブロックの場合は次の手順を実行します。
  - BeeGFS管理およびメタデータサービス用に2+2 RAID 10ボリュームグループが1つ

- BeeGFSストレージサービス用に8+2 RAID 6ボリュームグループが2つ

- BeeGFSメタデータとストレージビルディングブロックの場合：
  - BeeGFSメタデータサービス用に2+2 RAID 10ボリュームグループが1つ
  - BeeGFSストレージサービス用に8+2 RAID 6ボリュームグループが2つ
- BeeGFSストレージのみのビルディングブロックの場合：
  - BeeGFSストレージサービス向けに10+2 RAID 6ボリュームグループが2つ

> BeeGFSでは管理とメタデータのストレージ容量がストレージよりも大幅に少ないため、RAID 10ボリュームグループには小さいドライブを使用する方法も1つあります。より小さいドライブは、最も外側のドライブスロットに取り付ける必要があります。詳細については、を参照してください "導入手順"。

これらはすべてAnsibleベースの導入で設定され、次のようなパフォーマンスや動作を最適化するために一般的に推奨されるいくつかの設定とともに使用されます。

- グローバルキャッシュブロックサイズを32KiBに調整し、デマンドベースのキャッシュフラッシュを80%に調整しています。
- 自動ロードバランシングを無効にする（コントローラのボリュームの割り当ては意図したとおりに維持する）。
- 読み取りキャッシュの有効化と先読みキャッシュの無効化
- ミラーリングあり、バッテリバックアップを必要とする書き込みキャッシュを有効にして、ブロックノードコントローラの障害後もキャッシュを維持できるようにします。
- ボリュームグループにドライブを割り当てる順序を指定し、使用可能なドライブチャネル間でI/Oのバランスを調整します。

大容量のビルディングブロック

標準のBeeGFS解決策 設計は、ハイパフォーマンスのワークロードを念頭に置いて構築されます。大容量のユースケースを検討している場合は、ここで紹介するような設計やパフォーマンスの特徴の違いを確認する必要があります。

ハードウェアとソフトウェアの設定

大容量ビルディングブロック用のハードウェアとソフトウェアの設定は標準です。ただし、EF600コントローラをEF300コントローラに交換し、各ストレージアレイ用に60本のドライブを搭載した1 ~ 7個のIOM拡張トレイを取り付けることができます。ビルディングブロックあたり2～14台の拡張トレイを搭載。

大容量のビルディングブロック設計を導入する場合、使用されるのは、BeeGFSの管理、メタデータ、ストレージの各サービスで構成される基本ビルディングブロック形式の設定のみです。コスト効率を高めるために、大容量のストレージノードでは、EF300コントローラエンクロージャ内のNVMeドライブにメタデータボリュームをプロビジョニングし、拡張トレイ内のNL-SASドライブにストレージボリュームをプロビジョニングする必要があります。

[]

サイジングガイドライン

これらのサイジングガイドラインでは、大容量のビルディングブロックが、ベースEF300エンクロージャ内のメタデータ用に2+2のNVMe SSDボリュームグループ1つ、ストレージ用にIOM拡張トレイあたり6x8+2のNL-SASボリュームグループ1つで構成されていることを前提としています。

ドライブサイズ（大容量**HDD**）	**BB**あたりの容量**(1**トレイ**)**	**BB**あたりの容量（トレイ**×2**）	**BB**あたりの容量（トレイ**×3**）	**BB**あたりの容量**(4**トレイ**)**
4TB	439TB	878 TB	1、317 TB	1、1756 TB
8 TB	878 TB	1、1756 TB	2、634 TB	3、512TB
10 TB	1、097 TB	2195 TB	3、3292 TB	4390 TB
12TB	1、317 TB	2、634 TB	3951 TB	5268 TB
16 TB	1、1756 TB	3、512TB	5268 TB	7024 TB
18 TB	1975 TB	3951 TB	5927TB	7902 TB

著作権に関する情報

商標に関する情報