

# PostgreSQLのための 共有ディスク型スケールアウトの開発

2019年11月15日

富士通株式会社

データマネジメント事業部

綱川 貴之

## ■ コミュニティ市民

- PostgreSQL contributor
- PostgreSQL Ecosystem Wiki作成 & 維持者
- PostgreSQLエンタープライズ・コンソーシアム CR部会員

## ■ 企業人

- コミュニティ活動チームリーダー
- FUJITSU Software Enterprise Postgres (FEP)開発者

- 共有ディスク型スケールアウトの開発のきっかけ
- スケーラビリティ向上の選択肢の比較
- 共有ディスク型スケールアウトのアーキテクチャと設計
- 共有ディスク型スケールアウトは必要か？

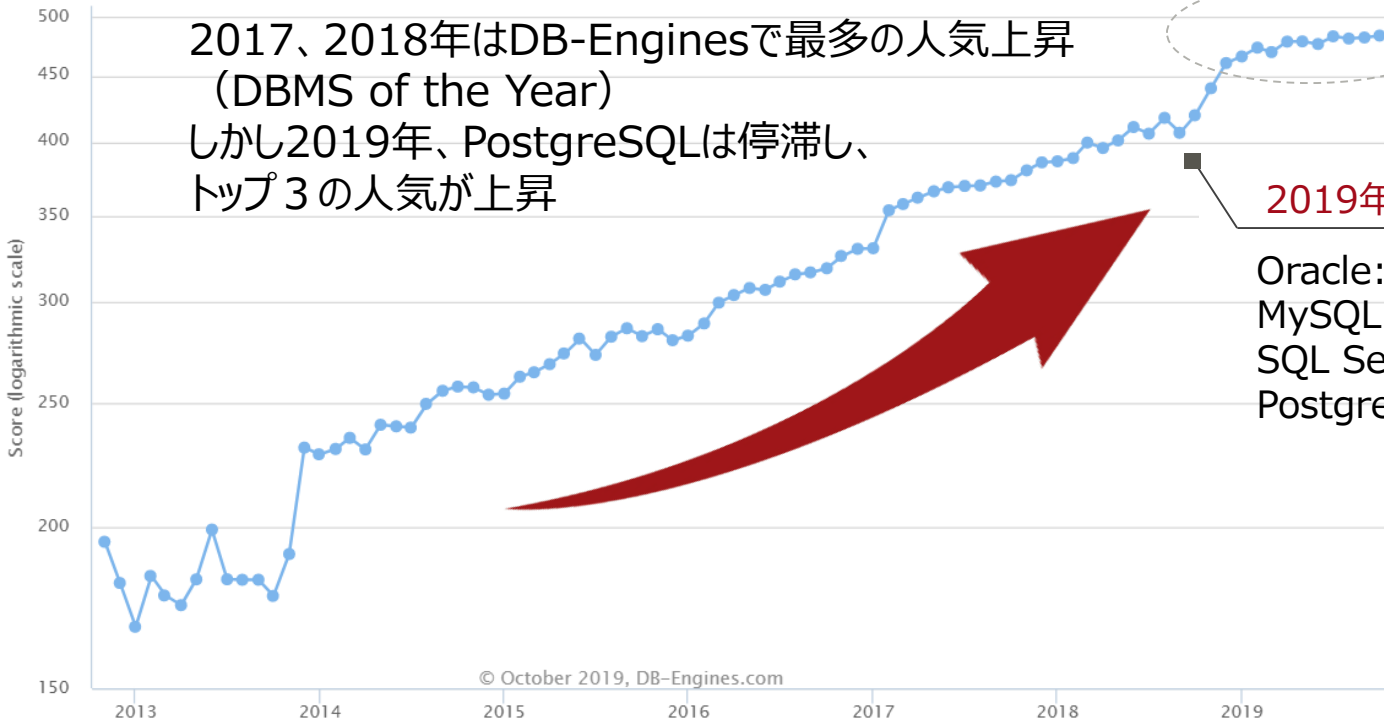
# PostgreSQLの人気に陰り？

DB-Engines Ranking of PostgreSQL

2017、2018年はDB-Enginesで最多の人気上昇  
(DBMS of the Year)  
しかし2019年、PostgreSQLは停滞し、  
トップ3の人気上昇

2019年のランキング

Oracle: 1269 → 1355, +86  
MySQL: 1154 → 1283, +129  
SQL Server: 1040 → 1095, +55  
PostgreSQL: 466 → 484, +18



© October 2019, DB-Engines.com

[https://db-engines.com/en/ranking\\_trend/system/PostgreSQL](https://db-engines.com/en/ranking_trend/system/PostgreSQL)

- AzureのVMの約半数がLinux
- トップ100万サイトの40%でLinuxが稼動、Windowsは33%  
(W3Techs.com, Oct 2019)
- サーバOS市場で成長率No. 1、売上シェア2位

# さらに人気を高めるには何をすべきか？

- スケールアウト: より多くのデータと要求を捌く
- ハードウェア活用で高速化: DRAMや不揮発メモリ、GPU、FPGA
- マルチモデル: 多様な形式のデータを扱う  
(キーバリュー、文書、グラフ、時系列...)
- より高いセキュリティ: 暗号化、権限分掌、SQLファイアウォール
- 他DBMSからの移行性向上



より大きく、速く、  
何でもできて便利で安全に

## ■ 「Oracle RAC相当の機能がほしい」

1. サーバ1台の処理能力を超えられるよう備えたい
  - Exadataの性能と拡張性に満足
2. コスト効率よく高い可用性を確保したい
  - 待機サーバを遊ばせず、データの読み書きに使いたい
3. アプリケーションを変更したくない
  - データの分割と配置は難しい

- メインフレーム/ミニコン時代から25年以上の積み重ねがある
  - Oracle Parallel Server (OPS) for VAX/VMS: 1990年代前半
    - Oracle Real Application Clusters (RAC): 2001年
  - IBM DB2 for z/OS: 1990年代前半
    - IBM DB2 pureScale for AIX, Linux: 2009年
  - Sybase Adaptive Server Enterprise Cluster Edition: 2010年
- この高度な技術をそう簡単にPostgreSQLに取り入れられるか？
- できたら有用か？ PostgreSQLの飛躍につながるか？



## 1. shared disk型スケラアウト (以降、SD)

- Oracle RAC, IBM Db2 pureScale

## 2. shared nothing型スケラアウト (以降、SN)

- Oracle Sharding, IBM Db2
- Google Cloud Spanner, CockroachDB, MySQL Cluster
- Azure Database for PostgreSQL - Hyperscale (Citus)
- Greenplum, Postgres-XL

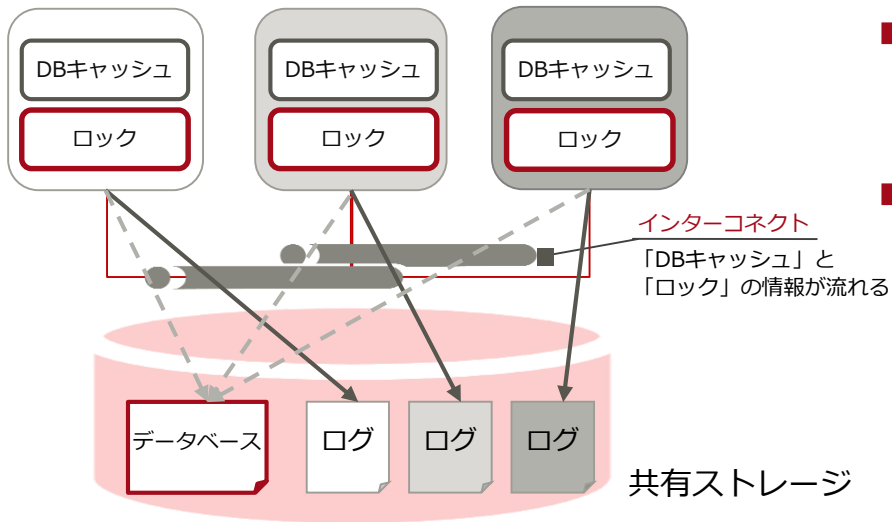
## 3. スケラアップ

# Shared disk方式の概要

アプリケーション  
/Webサーバ



インスタンス1 インスタンス2 インスタンス3



- 複数のDBインスタンスが共有ストレージ上のデータベースにアクセス
- DBキャッシュとロックは、DBインスタンス間で調整しながら分散管理
- 各DBインスタンスは個別のトランザクションログに書き込む

# Shared nothing方式の概要

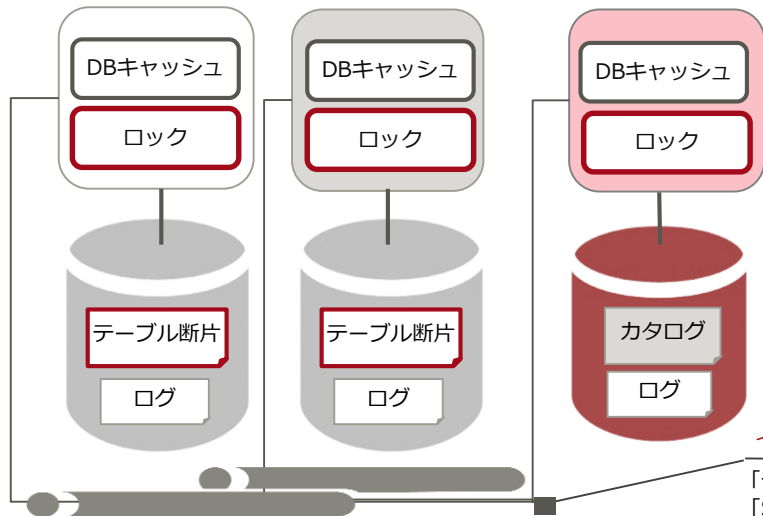
アプリケーション  
/Webサーバ



Worker

Worker

Manager



- 列の値によりテーブルを行集合に断片化し、断片をDBインスタンス群に分散して格納
- 各DBインスタンスは、ローカルストレージ上の自分のテーブル断片とトランザクションログだけ読み書き
- 各DBインスタンスは、自分のテーブル断片用のDBキャッシュとロックを管理

インターコネクト  
「テーブルデータ」と  
「SQL/実行計画」が流れる

- 処理能力の拡張性: SN > SD > Scaleup
  - 単一組織内のユーザ向けOLTPにはスケールアップで十分?
    - x86コモディティサーバ: 2ソケット/56コア, 数百GB RAM
    - X86ハイエンドサーバ (HPE Superdome) : 32ソケット/896コア, 48TB RAM
    - クラウドVM (Azure M208ms v2) : 208 vCPU, 5700 GB RAM
  - SD方式では、RACが100ノード、pureScaleが128ノードまで拡張
    - pureScaleは、読み取り大半のWebコマースのワークロードの場合に、
    - 64ノードで95%、128ノードで84%の高いスケーラビリティを発揮 (出典: IBM)
  - SN方式は競合を生じる共有資源が少ないため、スケーラビリティが最も高い

## ■ 性能比較

### ■ TPC-Cでは、SNもSDも高い性能を発揮

- 1位=SN 6,088万tpmC (Alibaba OceanBase, 2019年)
- 2位=SD: 3,024万tpmC (Oracle RAC, 2010年)
- 3位=SN: 1,036万tpmC (IBM DB2, 2010年)

## ■ 拡張の容易さ: Scaleup > SD > SN

- Scaleup: サーバにCPUやメモリを追加、VMインスタンスを変更
- SD: サーバを追加。性能のために物理データ構造の変更が必要なことも
- SN: サーバを追加した上で、データを再分散したり、アプリの接続先を変更

## ■ DBサーバ間の距離: SN > SD

- SD: データセット全体を共有するため、すべてのDBサーバを近い距離に配置
- SN: グローバル分散DBでは、国や地域ごとにデータセットを分割し、各地域のユーザに近いDBサーバがデータを所有

## ■ 負荷分散: SD > SN

- SN: DBサーバの負荷を均一にするには、アクセス頻度が同等になるようデータを分散

## ■ キャッシュの有効性: SN > SD > Scaleup

- SD: 各DBサーバはデータセット全体をキャッシュ対象にせねばならない
- SN: 各DBサーバは自身が保持するデータのみをキャッシュすればよい

## ■ ホットスポットへの耐性: SN > Scaleup > SD

一部のデータやトランザクションログへの読み書き集中をどう緩和できるか?

- SN: DBサーバ間でデータを分割し、アクセスを分散
- Scaleup: CPUやストレージを増強し、ホットスポットでの処理待ちを短縮
- SD: 読み込みのホットスポットはDBサーバの追加やストレージ増強で緩和  
書き込みがあると、キャッシュ整合性のために、  
同一ブロックでの競合時の待機時間が伸長

- データ注入と分析、バッチ処理: SN > SD > Scaleup
  - 大量データの処理にはSNが最適
  - 読み書きの応答時間を短くするには、データと処理を分割して並列に実行
  - スループットを上げるには、ボトルネックになる共有資源を減らす
  - TPC-Hの上位ランクを占めるのもSN



## ■ マルチテナントOLTP: SN > SD > Scaleup

ユーザやデバイス、組織や店ごとにデータと処理を分割できるアプリ

### ■ SN: テナントIDを含むキーでデータをDBサーバに分散

アプリはテナントIDを使って接続プールから接続を選ぶ

参照データのように分割できないものは、管理サーバに置くか、

すべてのDBサーバに複製

### ■ SD: SNと同様、データを分割し、あるテナントの要求を特定DBサーバに送る

共有ストレージがボトルネック。参照データを複製しなくてよいのが利点

### ■ SNもSDも、シーケンスがボトルネック

### ■ SNでの二次インデックスを使ったアクセスは、複数DBサーバにまたがり、遅くなりがち

## ■ シングルテナントOLTP: SD > Scaleup > SN

1つの組織のユーザが使う、データを分割しづらいアプリ

- Scaleup: コモディティサーバで処理できるなら、最もシンプルでコスト効率が高い
- SD: データはそのままで、サーバ（CPUとメモリ）を追加してスループットを向上  
バッファとロックの分散管理のため、応答時間は単一サーバより伸びる  
同じブロックに読み書きが集中すると、キャッシュ整合性のためサーバ間のデータ転送が増加し、性能が悪化  
例: インデックスへのキーの昇順での行挿入、シーケンスからの採番
- SN: DBサーバ間でのデータ転送や分散 2 PCにより、応答時間が伸びる

## ■ サーバ障害への耐性: $SD > SN = \text{Scaleup}$

- SN: 故障したサーバのデータのみアクセスできなくなる

サーバ台数が増えると、いずれかが故障する確率が高まり、クラスタ全体の可用性は低下  
管理ノードが単一障害点

- SD: どのDBサーバが故障しても、生存サーバがすべてのデータにアクセスできる

## ■ ストレージ障害への耐性: $SN = SD = \text{Scaleup}$

いずれの方式でも、ハードウェアかソフトウェアでの冗長化が必要

## ■ フェイルオーバの影響: $SD > SN = \text{Scaleup}$

- Scaleup, SN: DBサーバのリカバリ中は、それが保持するデータだけがアクセス不可

- SD: 故障したDBサーバが更新したデータ以外は、他のDBサーバからアクセス可能

RACでは、リマスターとリカバリ集合の特定の間は、クラスタ全体の活動がフリーズ

## ■ データ配置: SD = Scaleup > SN

- SN: ノード間転送と2PCの回避と負荷均衡のため、データの分割と配置を設計  
ユーザや店のIDでテーブルをパーティション化し、DBサーバ群に分散  
参照データのコピーをすべてのDBサーバに配置

## ■ ワークロード管理: Scaleup > SD > SN

- SN: アプリは必要なデータを持つDBサーバにトランザクション要求を発行
- SD: すべてのデータが共有されるため、データの配置先を意識する必要はない  
しかし、同一ブロックへの読み書きの競合を減らすように、  
DBサーバ間でワークロードを分割するのがよい

## ■ アプリケーション変更: Scaleup > SD > SN

- どの方式でも、DML文は変更不要
- SN: データ配置の設計に基づいてDDL文を変更
  - ハッシュや範囲でテーブルをパーティション化し、異なるDBサーバに分散
  - シーケンスのキャッシュを大きくしたり、順序性を強制しない
- SD: DBサーバ間での同一ブロックへの読み書き競合を減らすようDDL文を変更

## [RACでの推奨]

- 索引の数を減らす
- シーケンスのキャッシュを大きくしたり、順序性を強制しない
- ノード固有のシーケンス範囲値を生成する (スケーラブル・シーケンス)
- 小さなブロックサイズを選んだり、ブロックの空き領域を設定する
- プライマリキーの索引をパーティション化する
- テーブルをハッシュ・パーティション化してローカル索引を作成する
- キーのビット列を反転させた逆キー索引を使う

## ■ サーバ台数: Scaleup = SD > SN

- Scaleup, SD: データをアクセスするDBサーバだけが必要

- SN: 管理サーバとそのスタンバイが必要

クラスタ・メンバーシップやDBカタログ、シーケンス、トランザクションを管理

## ■ 待機サーバの余剰能力: SD > SN = Scaleup

- Scaleup, SN: 待機サーバで読み取りクエリやバックアップを実行できるが、書き込みはできない

- SD: 待機専用のサーバはなく、すべてのサーバが読み書きできる

## ■ ストレージ: Scaleup > SN > SD

- Scaleup: 直接接続ストレージ (DAS) を利用でき、  
ストレージの性能と容量を最大限に活用
- SN: Scaleupと同じだが、うまくデータと処理を分散できないと、容量と性能が過不足に
- SD: DASを利用できないため、価格性能比は下がる  
ネットワーク経由でストレージにアクセス (FC, NVMe-oF, iSCSI)  
分散ファイルシステムを使う場合は、さらにストレージサーバを経由 (NFS, Ceph)  
サーバのメモリスロットに装着する永続メモリは、真価を発揮する形では利用不可



# 方式比較のまとめ (1/2)

## ■ スケーラビリティと価格性能比は、SNがSDより高い

### スケーラビリティと性能

比較項目	SN	SD	Scaleup
処理能力の拡張性	3	2	1
拡張の容易さ	1	2	3
DBサーバ間の距離	3	1	1
負荷分散	1	3	1
キャッシュの有効性	3	2	1
ホットスポットへの耐性	3	1	2
データ注入と分析、バッチ	3	2	1
マルチテナントOLTP	3	2	1
シングルテナントOLTP	1	3	2
小計	21	18	13

### コスト

比較項目	SN	SD	Scaleup
サーバ台数	2	3	3
ストレージ	2	1	3
待機サーバの余剰能力	2	3	2
小計	6	7	8

## ■ 可用性とアプリケーション透過性は、SDがSNより優れる

### 可用性

比較項目	SN	SD	Scaleup
サーバ障害への耐性	1	3	1
ストレージ障害への耐性	1	1	1
フェイルオーバーの影響	1	2	1
小計	3	6	3

### アプリケーション透過性

比較項目	SN	SD	Scaleup
データ配置	1	3	3
ワークロード管理	1	2	3
アプリケーション変更	1	2	3
小計	3	7	9

# PostgreSQL向けShared disk方式の全体構成

アプリケーション  
/Webサーバ



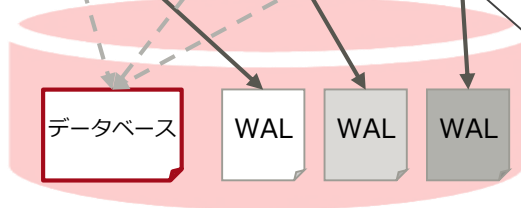
インスタンス1 インスタンス2 インスタンス3



Coordinator



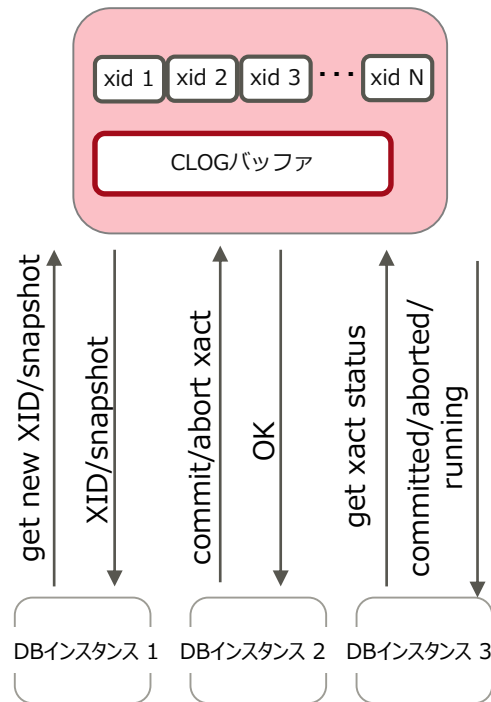
インターコネクト



「DBキャッシュ」、「ロック」と「XID/スナップショット」の情報が流れる

- 最大128のDBインスタンスのほかに、別サーバでCoordinatorが稼動
- Coordinatorはトランザクション、DBキャッシュ、ロックを集中管理
- SPoFを排除するため、Coordinatorはマスター-スタンバイ構成で動作
- ファイルシステムはクラスタFS (Red Hat GFS2, GPFS) か分散FS (Ceph, NFS)

Coordinator



- Coordinatorは次を管理
  - 動作中のトランザクションID (XID) 一覧
  - トランザクションの状態を示すコミットログ (CLOG)
- バックエンドからの要求に応じ、Coordinatorは
  - XIDを割り当てたり、スナップショットを作成し、依頼者に返却
  - トランザクションの完了をXID一覧とCLOGに記録
  - トランザクションの状態をCLOGで調べて依頼者に通知

Coordinator

B1	Modified	inst1
B2	Shared	inst1 inst2
B3	Shared	inst1 inst2

DBインスタンス1 DBインスタンス2 DBインスタンス3



- 個別にキャッシュを持つDBインスタンス群が同一ブロックを読み書き  
→ キャッシュ整合性が必要
- キャッシュ一貫性 ≡ 古いデータを読まないようにする
- CPUのキャッシュ一貫性プロトコルMESIにならう
  - 複数DBインスタンスが同一ブロックのクリーン (Shared) コピーを持てる
  - DBインスタンスが書き込んでブロックをダーティ (Modified) にすると、他のDBインスタンスが持つコピーは破棄される (Invalidate)
  - Coordinatorは各ブロックのキャッシュ先インスタンスと状態を管理し、やりとりを仲介

## ■ CoordinatorとDBインスタンスとで管理を分担

- Coordinatorはデッドロック検出のためにロックテーブルを集中管理
- しかし、常に全ロックを管理すると、Coordinatorがボトルネックに
- PG 9.2以降のFast Path Locking (FPL) により、DMLではCoordinatorとやりとりしない
- 弱ロック: SELECT, INSERT, DELETE, UPDATE
- 強ロック: ALTER/DROP, CLUSTER, REINDEXなど

## ■ DBインスタンス

- 弱ロックは従来どおりFPLで高速に処理
- 強ロックはCoordinatorに要求して処理

## ■ Coordinator

- 強ロック要求を受けると、競合する弱ロックを持つDBインスタンスにロック情報を送るよう指示
- デッドロックを検出し、DBインスタンスにトランザクションのアボートを指示

## ■ ネットワーク

- オーバヘッドを小さくするため、DBインスタンス間通信の多くにUDPを使用
- InfiniBandやRoCE (RDMA over Converged Ethernet) でさらに高速化  
Mellanoxの100Gb Ethernet/IB両用NICでも\$795、50GbE NICなら\$475  
Azure HシリーズVMでInifiBandとRDMAが利用可
- マルチキャストは使わない  
Oracle RACはマルチキャストを使うため、AWSやAzureなどのクラウドでは使えない

## ■ クラスタ

- 監視とフェイルオーバ、フェンシングなどHAの枠組みはクラスタソフトにゆだねる

## ■ リカバリ

- CoordinatorがDBインスタンスの故障を検知し、いずれかのDBインスタンスにリカバリを指示
- 同一ページへの複数DBインスタンスの更新を守ってWALを適用  
そのために、ページ世代番号をページヘッダとWALレコードに追加
- PITRは全DBインスタンスのWALをマージし、元の更新順序で適用

PG-CALSの性能（出典：中山 陽太郎, 2007年）

米Unisysの協力の下でユニアデックス（株）が調査研究の途中成果を公表

DBサーバ2台、pgbench、同時接続数100

既存PostgreSQL 1台に対してtpsは何倍か？

参照:更新比率7:3の場合で1.4倍、0:10で1.8倍

私たちの実装では、次の違いによりこの倍率を高めたい

- バッファを更新しても、Coordinatorにデータを送らない
- FPLに基づき、DML文ではCoordinatorとやりとりしない
- 低遅延のネットワークとRDMA



# 共有ディスク型スケールアウトは必要か？

前提: 分析やWebスケールOLTPにはSN

疑問: SDが必要なOLTPはあるか？

## ①スケールアップで処理しきれるか？

- コモディティサーバでも56コア、1.5 TBまでのDRAMを搭載可能
- ストレージはNVMe SSD、永続メモリで高速化
- Amazonが約7500のOracle DBを撤廃（2019/10/15）
- 移行先はRDS, Aurora, Redshift, DynamoDB, ElastiCache

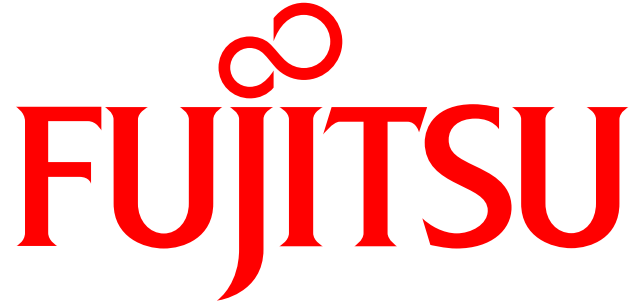
## ②SDはコスト効率のよいHAか？

- 故障サーバの処理を引き継ぐため、ふだん稼働サーバの使用率を抑える  
→ 待機サーバを遊ばせるのと何が違う？

## ③アプリケーションを変更してまでほしいか？変更できるか？

意見求む



The logo features a red infinity symbol positioned above the word "FUJITSU". The word "FUJITSU" is rendered in a bold, red, serif typeface. The letter "J" is stylized with a curved tail that extends downwards and to the left.

**FUJITSU**

shaping tomorrow with you