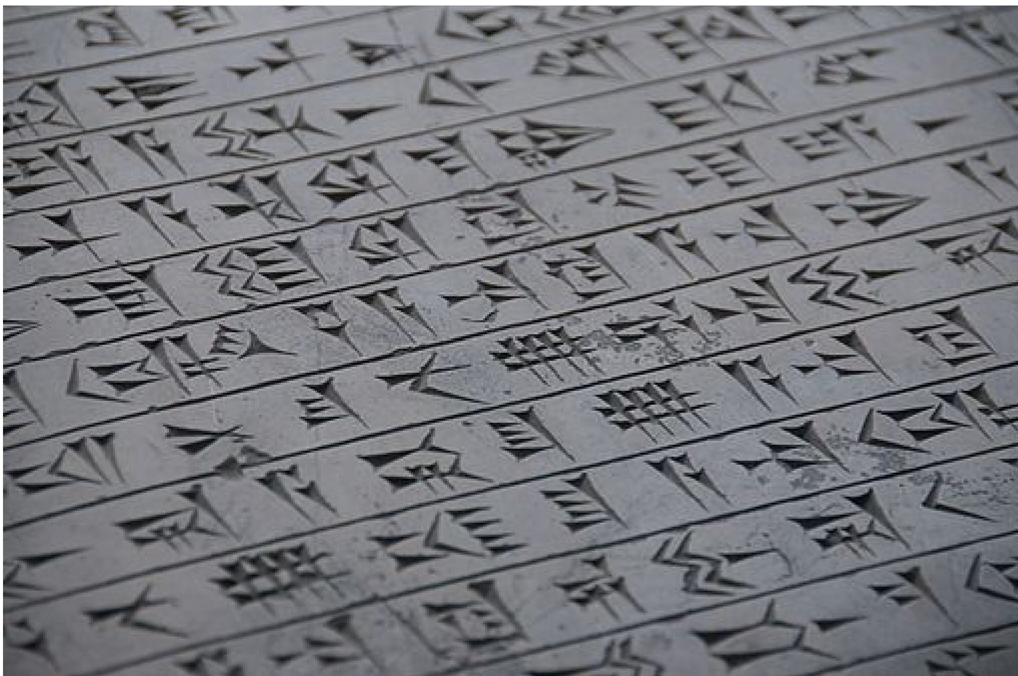
データ保持の概要

この記事は、インテル® デベロッパー・ゾーンに公開されている「Data Persistence in a Nutshell」の日本語参考訳です。

アプリケーションは、ファイルを使用して実行中のデータを保存します。これらのファイルにはいくつかの要件があります。

- プロセス終了後も存続すること
- 別のシステムにある別のプログラムによって読み取り可能であること
- バックアップおよび復元できること
- 複数のプロセスが同時にアクセス可能なこと
- OS がファイルの一部をキャッシュできること、スレッドが要求する前にディスクからファイルデータの読み取りを開始できること、およびファイルデータをディスクに書き込む間スレッドが処理を継続できること

大容量の不揮発性メモリーデバイスは、ディスクベースのファイルシステムよりも効率良くデータを格納できます。このようなメモリーデバイスの利点を得るためのプログラミングはさまざまで、上記のどの要件を満たす必要があるのかに依存します。



出典: Wikimedia Commons、米国オレゴン州ポートランド A. Davey 撮影 (Detail - Cuneiform Inscription) [CC BY 2.0 (http://creativecommons.org/licenses/by/2.0) の下に掲載]

SSD と HDD のレイテンシーと帯域幅のオーバーヘッドを排除する

ソリッド・ステート・ドライブ (SSD) やハード・ディスク・ドライブ (HDD) を不揮発性メモリーデバイスに交換することで、メディアとの間のデータ転送時間を大幅に排除できます。ただし、OS オーバーヘッドは排除されません。

ほとんどのアプリケーションではファイルの場所を指定できるため、この交換によりアプリケーションを変更する必要はありません。

実際のファイルシステムを使用して OS オーバーヘッドを排除する

スレッドがファイルをメモリーにマップする場合、ユーザー空間から OS への移行時間を軽減できます。ただし、ファイルの内容をインメモリー表現に変換する時間は変わりません。

SSD または HDD メモリーにマップされたファイルは、最初にタッチされたときにデータをロードするための遅延が発生します。不揮発性メモリーにマップされたファイルは、ページがメモリーにあるため、ページフォルトが発生しません。

データを変換することで、変換時間のオーバーヘッドを補うだけの利点が得られます。インファイル表現はインメモリー表現よりも圧縮可能であるため、デバイスにより多くのデータを格納できます。データにアクセスするアプリケーションは、独自のデータ構造を使用することができ、インファイル表現に影響を与えることなく変更可能です。最後に、OS とプログラミング言語のメモリー保護機能を利用することで、アプリケーションがインメモリー表現で書き込む場合であっても、インファイル表現の破損を防ぐことができます。

ほとんどの言語は、I/O 操作によりファイルではなく文字列の読み取り/書き込みをアプリケーションに許可しています。そのため、データ構造をメモリーに保持したまま、インファイル表現で格納できる文字列に簡単に変換できます。

変換オーバーヘッドを排除する

メモリーにマップされたデータをスレッドが使用する表現で表す場合、変換オーバーヘッドを排除できます。ただし、これにはマップ可能なデータ構造を意図的に制限することでのみ排除できるコストが伴います。

データ構造にメモリーアドレスが含まれる場合、ターゲット・エンティティーを同じアドレスに再ロードする必要があります。これは簡単なことではありません。アドレスは、ソースコードではなくコンパイラーによって生成された可能性があります (C++ の仮想関数の実装ではこれが行われます)。あるいは、ライブラリーによって生成された可能性もあります (C++ STL コンテナーはこれを行います)。ターゲット・エンティティーが別のファイルにあるか、プログラムコードの関数または vtable のアドレスである可能性もあります (C++ コンパイラーはこれを行うコードを生成します)。

設計の早期段階で、次の 1 つ以上の方針を選択する必要があります。

- エンティティーが適切な場所にロードされていることを確認し、そうでない場合はキャンセルします。
- インファイル表現のアドレスは使用しないようにします。
- 使用する前にアドレスを再配置します。

通常、インデックスを使用し、インデックスとアドレスの相対表を維持することで、アドレスの使用を回避することができます。

共有について

メモリーにマップされたファイルをプロセス間で共有することと、スレッド間でデータ構造を共有することに大きな違いはありません。どちらの場合も、メモリーアクセスの順序付けとキャッシング、クロスプロセス・ロック、アドレスの課題にコードで対応する必要があります。

フォールトトレランスについて

ハードウェアやアクセスしているソフトウェアの一部で致命的なエラーが発生した場合、データを保持するには次のことが必要です。

- ハードウェアの残りの部分に、データの一貫性のある表現が残されていること
- 残りのハードウェアをエラーが発生した部分から切り離すこと

どのようなエラーが発生したときにデータを保持するのかを決定し、ハードウェアとソフトウェアを使用して、エラーからデータを切り離すためのファイアウォールを作成する必要があります。例えば、エイリアンによる地球破壊からデータを守るには、どこか別の場所へデータを複製する必要があります。

多くのケースで重要なことは、次のような場合にどうすべきか明確にすることです。

- アプリケーションが誤ってデータを破損してしまった場合
- メモリーデバイスが利用できなくなった場合
- プロセスがキル (終了) された場合

アプリケーションによる過ちは、適切な動作とそうでない動作の区別が難しいため、最も困難なシナリオと言えます。このシナリオに対応するには、以前の状態を適時にバックアップする必要があります。同様に、メモリーデバイスの消失に対応する場合も、適時のバックアップが必要です。

残るは、最も簡単なシナリオである、アプリケーションが書き込みを停止した場合です。例えば、メモリーにマップされたファイルを変更中のアプリケーションがハードウェアのエラーによりメディアに書き込みできない場合、または一連の書き込み中にアプリケーションが中断された場合などです。コンパイラーとメモリー・サブシステムはどちらも書き込みの並べ替えを行い、書き込みはいつでも中断される可能性があるため、特定の順序で書き込みをコーディングするだけでは問題を回避できません。

フェンスまたはバリア命令 (英語) と、(特にデータベース分野において) 広く研究されているトランザクション (英語) の概念を使用することで、この問題に対応できます。Undo (元に戻す)/Redo (やり直し) ログを使用して、次回ファイルがマップされたときに失われた書き込みを補います。

まとめ

以前の記事、メモリー・パフォーマンスの概要では、揮発性メモリーの動作を説明しました。この記事では、不揮発性データを向上する方法を示しました。次の記事、NUMA システムを使用する場合のハードウェアとソフト

ウェアのアプローチ (英語) では、ここで紹介した手法を実装するため、アプリケーションを変更する手順を示します。

著者紹介

Bevin Brett。インテル コーポレーションの主任エンジニアで、プログラマーとシステムユーザーがアプリケーション・パフォーマンスを向上できるように支援するツールに取り組んでいます。3 人の娘 (長女は高校の数学教師、次女は産婦人科医、三女は絶滅危惧種の鳥の飼育員) を溺愛する父親です。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。