



Informatica® PowerCenter
10.4.0

パフォーマンスのチューニングの概要

米政府の権利プログラム、ソフトウェア、データベース、および関連文書や技術データは、米国政府の顧客に配信され、「商用コンピュータソフトウェア」または「商業技術データ」は、該当する連邦政府の取得規制と代理店固有の補足規定に基づきます。このように、使用、複製、開示、変更、および適応は、適用される政府の契約に規定されている制限およびライセンス条項に従うものとし、政府契約の条項によって適当な範囲において、FAR 52.227-19、商用コンピュータソフトウェアライセンスの追加権利を規定します。

本製品には、Apache Software Foundation (<http://www.apache.org/>) によって開発されたソフトウェア、およびさまざまなバージョンの Apache License（まとめて「License」と呼んでいます）の下に許諾された他のソフトウェアが含まれます。これらのライセンスのコピーは、<http://www.apache.org/licenses/> で入手できます。適用法にて要求されない書面に合意されない限り、ライセンスの下に配布されるソフトウェアは「現状のまま」で配布され、明示的あるいは黙示的の保証を問わず、いかなる種類の保証や条件も付帯することはありません。ライセンス下での許諾および制限を定める具体的文言については、ライセンスを参照してください。

製品には、ワシントン大学、カリフォルニア大学アーバイン校、およびバンダービルト大学の Douglas C.Schmidt および同氏のリサーチグループが著作権を持つ ACE (TM) および TAO (TM) ソフトウェアが含まれています。Copyright (C) 1993-2006, All rights reserved.

本製品には、Curl ソフトウェア Copyright 1996-2013, Daniel Stenberg, <daniel@haxx.se>が含まれます。All rights reserved. 本ソフトウェアに関する許諾および制限は、<http://curl.haxx.se/docs/copyright.html> にある使用条件に従います。すべてのコピーに上記の著作権情報とこの許諾情報が記載されている場合、目的に応じて、本ソフトウェアの使用、コピー、変更、ならびに配布が有償または無償で許可されます。

本製品には、Per Bothner のソフトウェアが含まれます。Copyright (C) 1996-2006.All rights reserved. お客様がこのようなソフトウェアを使用するための権利は、ライセンスで規定されています。<http://www.gnu.org/software/kawa/Software-License.html> を参照してください。

本製品には、Boost (<http://www.boost.org/>) によって開発されたソフトウェア、または Boost ソフトウェアライセンスの下で開発されたソフトウェアが含まれます。本ソフトウェアに関する許諾および制限は、http://www.boost.org/LICENSE_1_0.txt にある使用条件に従います。

本製品には、The Eclipse Foundationのソフトウェアが含まれます。Copyright (C) 2007.All rights reserved. 本ソフトウェアに関する許諾および制限は、<http://www.eclipse.org/org/documents/epl-v10.php> および <http://www.eclipse.org/org/documents/edl-v10.php> にある使用条件に従います。

本製品には、<http://www.tcl.tk/software/tcltk/license.html>、<http://www.bosrup.com/web/overlib?License>、<http://www.stlport.org/doc/license.html>、<http://www.asm-ow2.org/license.html>、<http://www.cryptix.org/LICENSE.TXT>、<http://hsqldb.org/web/hsqllicense.html>、<http://htpunit.sourceforge.net/doc/license.html>、<http://jung.sourceforge.net/license.txt>、http://www.gzip.org/zlib_license.html、<http://www.openldap.org/software/release/license.html>、<http://www.libssh2.org/http://slf4j.org/license.html>、<http://www.sente.ch/software/OpenSourceLicense.html>、<http://fusesource.com/downloads/licenses-agreements/fuse-message-broker-v-5-3-license-agreement>、<http://antlr.org/license.html>、<http://aopalliance.sourceforge.net/>、<http://www.bouncycastle.org/licenses.html>、<http://www.jgraph.com/jgraphdownload.html>、<http://www.jcraft.com/jsch/LICENSE.txt>、http://jotm.objectweb.org/bsd_license.html に基づいて許諾されたソフトウェアが含まれています。<http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>、<http://www.slf4j.org/license.html>、<http://nanoxml.sourceforge.net/orig/copyright.html>、<http://www.json.org/license.html>、<http://forge.ow2.org/projects/javaservice/>、<http://www.postgresql.org/about/license.html>、<http://www.sqlite.org/copyright.html>、<http://www.tcl.tk/software/tcltk/license.html>、<http://www.jaxen.org/faq.html>、<http://www.idom.org/docs/faq.html>、<http://www.slf4j.org/license.html>、<http://www.iodbc.org/dataspace/iodbc/wiki/iODBC/License>、<http://www.keplerproject.org/md5/license.html>、<http://www.toedter.com/en/calendar/license.html>、<http://www.edankert.com/bounce/index.html>、<http://www.net-snmp.org/about/license.html>、<http://www.forty-two.org/licenses.html>

www.openmdx.org/#FAQ、http://www.php.net/license/3_01.txt、<http://srp.stanford.edu/license.txt>、<http://www.schneier.com/blowfish.html>、<http://www.jmock.org/license.html>、<http://xsom.java.net>、<http://benalman.com/about/license/>、<https://github.com/CreateJS/EaselJS/blob/master/src/easeljs/display/Bitmap.js>、<http://www.h2database.com/html/license.html#summary>、<http://jsoncpp.sourceforge.net/LICENSE>、<http://jdbc.postgresql.org/license.html>、<http://protobuf.googlecode.com/svn/trunk/src/google/protobuf/descriptor.proto>、<https://github.com/rantav/hector/blob/master/LICENSE>、<http://web.mit.edu/Kerberos/krb5-current/doc/mitK5license.html>、<http://jibx.sourceforge.net/jibx-license.html>、<https://github.com/lyokato/libgeohash/blob/master/LICENSE>、<https://github.com/hjiang/jsonxx/blob/master/LICENSE>、<https://code.google.com/p/lz4/>、<https://github.com/jedisct1/libsodium/blob/master/LICENSE>、<http://one-jar.sourceforge.net/index.php?page=documents&file=license>、<https://github.com/EsotericSoftware/kryo/blob/master/license.txt>、<http://www.scala-lang.org/license.html>、<https://github.com/tinkerpop/blueprints/blob/master/LICENSE.txt>、<http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html>、<https://aws.amazon.com/asl/>、<https://github.com/twbs/bootstrap/blob/master/LICENSE>、および <https://sourceforge.net/p/xmlunit/code/HEAD/tree/trunk/LICENSE.txt>。

本製品には、Academic Free License (<http://www.opensource.org/licenses/afl-3.0.php>)、Common Development and Distribution License (<http://www.opensource.org/licenses/cddl1.php>)、Common Public License (<http://www.opensource.org/licenses/cpl1.0.php>)、Sun Binary Code License Agreement Supplemental License Terms、BSD License (<http://www.opensource.org/licenses/bsd-license.php>)、BSD License (<http://opensource.org/licenses/BSD-3-Clause>)、MIT License (<http://www.opensource.org/licenses/mit-license.php>)、Artistic License (<http://www.opensource.org/licenses/artistic-license-1.0>)、Initial Developer's Public License Version 1.0 (<http://www.firebirdsql.org/en/initial-developer-s-public-license-version-1-0/>) に基づいて許諾されたソフトウェアが含まれています。

本製品には、ソフトウェア copyright (C) 2003-2006 Joe Walnes, 2006-2007 XStream Committers が含まれています。All rights reserved. 本ソフトウェアに関する許諾および制限は、<http://j.org/license.html> にある使用条件に従います。本製品には、Indiana University Extreme! Lab によって開発されたソフトウェアが含まれています。詳細については、<http://www.extreme.indiana.edu/>を参照してください。

本製品には、ソフトウェア Copyright (C) 2013 Frank Balluffi and Markus Moeller が含まれています。All rights reserved. 本ソフトウェアに関する許諾および制限は、MIT ライセンスの使用条件に従います。

特許については、<https://www.informatica.com/legal/patents.html> を参照してください。

免責: 本文書は、一切の保証を伴わない「現状渡し」で提供されるものとし、Informatica LLC は他社の権利の非侵害、市場性および特定の目的への適合性の黙示の保証などを含めて、一切の明示的および黙示的保証の責任を負いません。Informatica LLC では、本ソフトウェアまたはドキュメントに誤りのないことを保証していません。本ソフトウェアまたはドキュメントに記載されている情報には、技術的に正確な記述や誤植が含まれる場合があります。本ソフトウェアまたはドキュメントの情報は、予告なしに変更されることがあります。

NOTICES

この Informatica 製品（以下「ソフトウェア」）には、Progress Software Corporation（以下「DataDirect」）の事業子会社である DataDirect Technologies からの特定のドライバ（以下「DataDirect ドライバ」）が含まれています。DataDirect ドライバには、次の用語および条件が適用されます。

1. DataDirect ドライバは、特定物として現存するままの状態を提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。
2. DataDirect または第三者は、予見の有無を問わず発生した ODBC ドライバの使用に関するいかなる直接的、間接的、偶発的、特別、あるいは結果的損害に対して責任を負わないものとします。本制限事項は、すべての訴訟原因に適用されます。訴訟原因には、契約違反、保証違反、過失、厳格責任、詐称、その他の不法行為を含みますが、これらに限るものではありません。

本マニュアルの情報は、予告なしに変更されることがあります。このドキュメントで問題が見つかった場合は、infa_documentation@informatica.com までご報告ください。

Informatica 製品は、それらが提供される契約の条件に従って保証されます。Informatica は、商品性、特定目的への適合性、非侵害性の保証等を含めて、明示的または黙示的ないかなる種類の保証をせず、本マニュアルの情報を「現状のまま」提供するものとします。

発行日: 2020-02-04

目次

序文	9
Informatica のリソース	9
Informatica Network	9
Informatica ナレッジベース	9
Informatica マニュアル	9
Informatica 製品可用性マトリックス	10
Informatica Velocity	10
Informatica Marketplace	10
Informatica グローバルカスタマサポート	10
第 1 章 : パフォーマンスのチューニングの概要	11
パフォーマンスのチューニングの概要	11
第 2 章 : ボトルネック	12
ボトルネックの概要	12
スレッド統計の使用	13
スレッド統計に基づくボトルネックの除去	13
例	13
ターゲットのボトルネック	14
ターゲットのボトルネックの特定	14
ターゲットのボトルネックの除去	14
ソースのボトルネック	15
ソースのボトルネックの特定	15
ソースのボトルネックの除去	16
マッピングのボトルネック	16
マッピングのボトルネックの特定	16
マッピングのボトルネックの除去	16
セッションのボトルネック	17
セッションのボトルネックの特定	17
セッションのボトルネックの除去	17
システムのボトルネック	17
システムのボトルネックの特定	17
システムのボトルネックの除去	18
第 3 章 : ターゲットの最適化	20
フラットファイルターゲットの最適化	20
インデックスおよびキー制約の削除	20
データベースのチェックポイント間隔を長く設定	21
バルクロードの使用	21
外部ローダーの使用	21

デッドロックの最小化.	22
データベースのネットワークパケットサイズの拡張.	22
Oracle ターゲットデータベースの最適化.	22
第 4 章 : ソースの最適化.	24
クエリの最適化.	24
条件フィルタの使用.	25
データベースのネットワークパケットサイズの拡張.	25
Oracle データベースソースへの接続.	25
Teradata FastExport の使用.	25
Sybase または Microsoft SQL Server のテーブルを結合するための tempdb の使用.	26
第 5 章 : マッピングの最適化.	27
マッピングの最適化の概要.	27
フラットファイルソースの最適化.	27
連続行のバッファ長の最適化.	28
区切りフラットファイルソースの最適化.	28
XML およびフラットファイルソースの最適化.	28
Single-Pass 読み込みの設定.	28
パススルーマッピングの最適化.	29
フィルタの最適化.	29
データタイプ変換の最適化.	30
式の最適化.	30
共通ロジックの抽出.	30
集計関数呼び出しの最小化.	30
ローカル変数での共通の式の置き換え.	31
文字列演算の代わりに数値演算を実行.	31
Char 対 Char および Char 対 Varchar の比較の最適化.	31
LOOKUP の代わりに DECODE を選択.	31
関数の代わりに演算子を使用.	31
IIF 関数の最適化.	31
式の評価.	32
External Procedure の最適化.	32
第 6 章 : トランスフォーメーションの最適化.	33
Aggregator トランスフォーメーションの最適化.	33
単純なカラム別のグループ化.	34
ソート済み入力の使用.	34
差分集計の使用.	34
集計前のデータのフィルタリング.	34
ポート接続の制限.	34
Custom トランスフォーメーションの最適化.	35
Joiner トランスフォーメーションの最適化.	35

Lookup トランスフォーメーションの最適化.	36
最適なデータベースドライバの使用.	36
ルックアップテーブルのキャッシュ.	36
ルックアップ条件の最適化.	38
Lookup 行のフィルタリング.	38
ルックアップテーブルのインデックス処理.	38
複数のルックアップの最適化.	38
パイプライン Lookup トランスフォーメーションの作成.	39
ノーマライズトランスフォーメーションの最適化.	39
Sequence Generator トランスフォーメーションの最適化.	39
Sorter トランスフォーメーションの最適化.	40
メモリの割り当て.	40
パーティションのワークディレクトリ.	40
Unicode モード.	40
Source Qualifier トランスフォーメーションの最適化.	41
SQL トランスフォーメーションの最適化.	41
XML トランスフォーメーションの最適化.	41
トランスフォーメーションエラーの除去.	42
第 7 章 : セッションの最適化.	43
グリッド.	43
ブッシュダウンの最適化.	44
コンカレントセッションおよびワークフロー.	44
バッファメモリ.	44
DTM バッファサイズの拡大.	45
バッファブロックサイズの最適化.	45
キャッシュ.	46
接続されるポート数の制限.	46
キャッシュディレクトリの場所.	46
キャッシュサイズの拡張.	47
64 ビットバージョンの PowerCenter の使用.	47
ターゲットベースのコミット.	47
リアルタイム処理.	48
フラッシュ待ち時間.	48
ソースベースのコミット.	48
ステージングエリア.	48
ログファイル.	48
エラートレース.	48
セッション実行後のメール.	49
第 8 章 : グリッドのデプロイメントの最適化.	50
グリッドのデプロイメントの最適化の概要.	50
ファイルの格納.	50

高帯域幅共有ファイルシステムのファイル.	51
低帯域幅共有ファイルシステムのファイル.	51
ローカルストレージファイル.	51
共有ファイルシステムの使用.	52
共有ファイルシステムの設定.	52
CPU とメモリの使用量の均衡化.	52
PowerCenter マッピングとセッションの設定.	53
ファイルシステム全体へのファイルの分散.	54
ファイルを分散するようにセッションを設定.	54
Sequence Generator トランスフォーメーションの最適化.	55

第 9 章 : PowerCenter コンポーネントの最適化. 57

PowerCenter コンポーネントの最適化の概要.	57
PowerCenter リポジトリのパフォーマンスの最適化.	57
リポジトリサービスプロセスおよびリポジトリの場所.	58
オブジェクトクエリの条件の整理.	58
単一ノードの DB2 データベースのテーブルスペース.	58
データベーススキーマの最適化.	58
リポジトリサービスにおけるオブジェクトキャッシュ.	59
レジリエンスの最適化.	59
Integration Service のパフォーマンスの最適化.	60
ネイティブドライバおよび ODBC ドライバの使用.	60
ASCII データ移動モードでの Integration Service の実行.	60
リポジトリサービスの PowerCenter メタデータのキャッシュ.	60

第 10 章 : システムの最適化. 61

システムの最適化の概要.	61
ネットワークの速度の向上.	62
複数の CPU の使用.	62
ページングの削減.	62
プロセッサバインドの使用.	63

第 11 章 : パイプラインパーティションの使用. 64

パイプラインパーティションの使用の概要.	64
パーティション数の増加.	64
最良のパフォーマンスを得るためのパーティションタイプの選択.	65
複数の CPU の使用.	66
パーティション化のためのソースデータベースの最適化.	66
データベースのチューニング.	67
ソート済みデータのグルーブ化.	67
単一のソート済みクエリの最適化.	67
パーティション化のためのターゲットデータベースの最適化.	68

付録 A: パフォーマンスカウンタ	69
パフォーマンスカウンタの概要.....	69
Errorrows カウンタ.....	69
Readfromcache カウンタおよび Writetocache カウンタ.....	70
Readfromdisk および Writetodisk カウンタ.....	70
Rowsinlookupcache カウンタ.....	71
索引	72

序文

*PowerCenter(R)*パフォーマンスチューニングガイドは、ランタイムのボトルネックの詳細と最適なパフォーマンスを得るチューニング方法を知るために参照します。

Informatica のリソース

Informatica は、Informatica Network やその他のオンラインポータルを通じてさまざまな製品リソースを提供しています。リソースを使用して Informatica 製品とソリューションを最大限に活用し、その他の Informatica ユーザーや各分野の専門家から知見を得ることができます。

Informatica Network

Informatica Network は、Informatica ナレッジベースや Informatica グローバルカスタマサポートなど、多くのリソースへの入口です。Informatica Network を利用するには、<https://network.informatica.com> にアクセスしてください。

Informatica Network メンバーは、次のオプションを利用できます。

- ナレッジベースで製品リソースを検索できます。
- 製品の提供情報を表示できます。
- サポートケースを作成して確認できます。
- 最寄りの Informatica ユーザーグループネットワークを検索して、他のユーザーと共同作業を行えます。

Informatica ナレッジベース

Informatica ナレッジベースを使用して、ハウツー記事、ベストプラクティス、よくある質問に対する回答など、製品リソースを見つけることができます。

ナレッジベースを検索するには、<https://search.informatica.com> にアクセスしてください。ナレッジベースに関する質問、コメント、ご意見の連絡先は、Informatica ナレッジベースチーム (KB_Feedback@informatica.com) です。

Informatica マニュアル

Informatica マニュアルポータルでは、最新および最近の製品リリースに関するドキュメントの膨大なライブラリを参照できます。マニュアルポータルを利用するには、<https://docs.informatica.com> にアクセスしてください。

製品マニュアルに関する質問、コメント、ご意見については、Informatica マニュアルチーム (infa_documentation@informatica.com) までご連絡ください。

Informatica 製品可用性マトリックス

製品可用性マトリックス（PAM）には、製品リリースでサポートされるオペレーティングシステム、データベースなどのデータソースおよびターゲットが示されています。Informatica PAM は、<https://network.informatica.com/community/informatica-network/product-availability-matrices> で参照できます。

Informatica Velocity

Informatica Velocity は、Informatica プロフェッショナルサービスが開発したヒントとベストプラクティスのコレクションで、多数のデータ管理プロジェクトから得た実体験に基づいています。Informatica Velocity には、世界中の組織と連携してデータ管理ソリューションを計画、開発、デプロイ、管理する Informatica コンサルタントによる集合知を表しています。

Informatica Velocity リソースには、<http://velocity.informatica.com> からアクセスしてください。Informatica Velocity についての質問、コメント、またはアイデアがある場合は、ips@informatica.com から Informatica プロフェッショナルサービスにお問い合わせください。

Informatica Marketplace

Informatica Marketplace は、お使いの Informatica 製品を拡張したり強化したりするソリューションを検索できるフォーラムです。Marketplace で、Informatica デベロッパーやパートナーからの多数のソリューションを活用すれば、生産性を向上したり、プロジェクトでの実装時間を短縮したりできます。Informatica Marketplace は、<https://marketplace.informatica.com> からアクセスしてください。

Informatica グローバルカスタマサポート

電話または Informatica Network からグローバルサポートセンターに連絡できます。

各地域の Informatica グローバルカスタマサポートの電話番号は、Informatica Web サイト（<https://www.informatica.com/services-and-training/customer-success-services/contact-us.html>）を参照してください。

Informatica Network でオンラインサポートリソースを見つけるには、<https://network.informatica.com> にアクセスし、eSupport オプションを選択します。

第 1 章

パフォーマンスのチューニングの概要

- [パフォーマンスのチューニングの概要, 11 ページ](#)

パフォーマンスのチューニングの概要

パフォーマンスのチューニングの目的は、パフォーマンスのボトルネックを取り除くことによって、セッションのパフォーマンスを最適化することです。セッションのパフォーマンスをチューニングするには、まずパフォーマンスのボトルネックを特定して、これを取り除きます。以後、満足できる水準のセッションパフォーマンスが得られるまで、ボトルネックの特定と除去を続けます。セッションのパフォーマンスをチューニングするときには、テストロードオプションを使用してセッションを実行できます。

すべてのボトルネックのチューニングが済むと、セッション内のパイプラインパーティションの数を増やして、セッションをさらに最適化することができます。パーティションを追加すると、セッションの処理中により多くのシステムハードウェアを利用できるようになるため、パフォーマンスが向上します。

パフォーマンスを向上させる最善の方法を見つけるには、複雑な作業が必要になります。そこで、変数を1つずつ変更し、その前後でのそれぞれのセッションの実行時間を比較します。その結果としてセッションのパフォーマンスが向上しなければ、設定を元に戻します。

セッションのパフォーマンスを改善するには、以下のタスクを実行します。

1. **ターゲットの最適化。** Integration Service による、ターゲットへの書き込みを効率化します。
2. **ソースの最適化。** Integration Service による、ソースデータの読み込みを効率化します。
3. **マッピングの最適化。** Integration Service による、データのトランスフォーメーションおよび移動を効率化します。
4. **トランスフォーメーションの最適化。** Integration Service による、マッピング内のトランスフォーメーションの処理を効率化します。
5. **セッションの最適化。** Integration Service による、セッションの実行速度を速くします。
6. **グリッドのデプロイメントの最適化。** Integration Service による、最適なパフォーマンスをグリッド上で実行可能にします。
7. **PowerCenter コンポーネントの最適化。** Integration Service およびリポジトリサービスによる、最適な機能を可能にします。
8. **システムの最適化。** PowerCenter サービスのプロセスによる、より高速での実行を可能にします。

第 2 章

ボトルネック

この章では、以下の項目について説明します。

- [ボトルネックの概要, 12 ページ](#)
- [スレッド統計の使用, 13 ページ](#)
- [ターゲットのボトルネック, 14 ページ](#)
- [ソースのボトルネック, 15 ページ](#)
- [マッピングのボトルネック, 16 ページ](#)
- [セッションのボトルネック, 17 ページ](#)
- [システムのボトルネック, 17 ページ](#)

ボトルネックの概要

パフォーマンスをチューニングする上での最初の手順として、パフォーマンスのボトルネックを特定します。パフォーマンスのボトルネックは、ソースおよびターゲットデータベース、マッピング、セッション、ならびにシステムで発生します。方針としては、パフォーマンスのボトルネックを特定して取り除いたら、以後、満足できる水準のパフォーマンスが得られるまでボトルネックの特定と除去を続けるというものです。

パフォーマンスのボトルネックは、以下の順に調査します。

1. ターゲット
2. ソース
3. マッピング
4. セッション
5. システム

次の方法でパフォーマンスのボトルネックを特定します。

- **テストセッションの実行。**フラットファイルソースからの読み込みを行うテストセッションまたはフラットファイルターゲットへの書き込みを行うテストセッションの際に、ソースおよびターゲットのボトルネックが特定されるようにテストセッションを設定できます。
- **パフォーマンスの詳細の分析。**パフォーマンスカウンタなどのパフォーマンスの詳細を分析して、セッションのパフォーマンスが低下している箇所を特定します。
- **スレッド統計の分析。**スレッド統計を分析して、最適なパーティションポイント数を決定します。
- **システムパフォーマンスの監視。**システム監視ツールを使用して、CPU 使用率、入出力待ちおよびページングを表示することにより、システムのボトルネックを特定できます。Workflow Monitor を使用して、システムリソースの使用状況を表示することもできます。

スレッド統計の使用

セッションログでスレッド統計を使用すると、ソース、ターゲットまたはトランスフォーメーションのボトルネックを特定できます。デフォルトでは、Integration Service はセッションを処理するために reader スレッドを 1 つ、トランスフォーメーションスレッドを 1 つ、writer スレッドを 1 つ使用します。ビジー率の最も高いスレッドが、セッションのボトルネックです。

セッションログは次のスレッド統計を示します。

- **実行時間。**スレッドの実行時間。
- **アイドル時間。**スレッドのアイドル時間。これには、アプリケーション内の他のスレッドの処理を、このスレッドが待機している時間が含まれます。アイドル時間には、Integration Service によってスレッドがブロックされた時間が含まれますが、オペレーティングシステムによってブロックされた時間は含まれません。
- **ビジー時間。**以下の式に従って計算されたスレッド実行時間の割合。

$$(\text{run time} - \text{idle time}) / \text{run time} \times 100$$

総実行時間が 60 秒以下などと短い場合、高いビジー率は無視できます。これは必ずしもボトルネックを示していません。

- **スレッド作業時間。**Integration Service で、1 つのスレッド内の各トランスフォーメーションの処理にかかった時間の割合。セッションログには、トランスフォーメーションスレッド作業時間に関する以下の情報が表示されます。

Thread work time breakdown:
 <transformation name>: <number> percent
 <transformation name>: <number> percent
 <transformation name>: <number> percent

トランスフォーメーションにかかった時間が短い場合、そのトランスフォーメーションはセッションログに記録されません。セッションの実行時間が短かったため、スレッドの正確な統計がない場合は、統計が正確でないことがセッションログにレポートされます。

スレッド統計に基づくボトルネックの除去

スレッド統計に基づいてボトルネックを除去するには、以下のタスクを実行します。

- reader または writer スレッドが 100%ビジーである場合、ソースポートまたはターゲットポートにて String データ型の使用を検討する必要があります。文字列でないポートは、さらに処理をする必要があります。
- トランスフォーメーションスレッドが 100%ビジーである場合は、セグメントへのパーティションポイントの追加を検討します。マッピングにパーティションポイントを追加すると、Integration Service はセッションで使用するトランスフォーメーションスレッドの数を増加します。ただし、マシンが最大限または最大限に近い容量で実行されている場合は、さらにスレッドを追加しないでください。
- 1 つのトランスフォーメーションが他より多くの処理時間を必要とする場合は、このトランスフォーメーションにパススルーパーティションポイントを追加することを検討します。

例

セッションを実行すると、セッションログは実行情報や、次のテキストと類似したスレッド統計を一覧表示します。

```
***** RUN INFO FOR TGT LOAD ORDER GROUP [1], CONCURRENT SET [1] *****
Thread [READER_1_1_1] created for [the read stage] of partition point [SQ_two_gig_file_32B_rows] has completed.
    Total Run Time = [505.871140] secs
    Total Idle Time = [457.038313] secs
    Busy Percentage = [9.653215]
Thread [TRANSF_1_1_1] created for [the transformation stage] of partition point [SQ_two_gig_file_32B_rows] has
```

```
completed.
  Total Run Time = [506.230461] secs
  Total Idle Time = [1.390318] secs
  Busy Percentage = [99.725359]
  Thread work time breakdown:
    LKP_ADDRESS: 25.000000 percent
    SRT_ADDRESS: 21.551724 percent
    RTR_ZIP_CODE: 53.448276 percent
Thread [WRITER_1_*_1] created for [the write stage] of partition point [scratch_out_32B] has completed.
  Total Run Time = [507.027212] secs
  Total Idle Time = [384.632435] secs
  Busy Percentage = [24.139686]
```

このセッションログでは、トランスフォーメーションスレッドの総実行時間が 506 秒、ビジー率が 99.7% です。これは、トランスフォーメーションスレッドが 506 秒の間、アイドルではなかったことを示します。reader および writer のビジーの割合は、それぞれ 9.6% および 24% と著しく低くなっています。このセッションでは、トランスフォーメーションスレッドがマッピングのボトルネックとなっています。

トランスフォーメーションスレッド内でボトルネックになっているトランスフォーメーションを特定するには、スレッド作業時間分析で各トランスフォーメーションのビジー率を確認します。このセッションログで、トランスフォーメーション RTR_ZIP_CODE のビジー率は 53% です。

ターゲットのボトルネック

最も一般的なパフォーマンスのボトルネックは、Integration Service がターゲットデータベースへの書き込みを行う場合に発生します。チェックポイント間隔が短いこと、データベースのネットワークパケットサイズが小さいこと、または負荷の高いロード操作時の問題によって、ターゲットのボトルネックが発生する可能性があります。

ターゲットのボトルネックの特定

ターゲットのボトルネックを特定するには、以下のタスクを実行します。

- フラットファイルターゲットに書き込むセッションのコピーを設定します。セッションのパフォーマンスが著しく向上する場合は、ターゲットにボトルネックがあります。そのセッションで既にフラットファイルターゲットに書き込みしている場合、ターゲットにはボトルネックがないと考えられます。
- セッションログのスレッド統計を読み込みます。Integration Service がトランスフォーメーションまたは reader スレッドよりも writer スレッドで時間を多く使用していれば、それがターゲットのボトルネックです。

ターゲットのボトルネックの除去

ターゲットのボトルネックを除去するには、以下のタスクを実行します。

- データベース管理者に、クエリーを最適化することによってデータベースのパフォーマンスを最適化するように依頼します。
- データベースのネットワークパケットサイズを大きくします。
- インデックスおよびキー制約を設定します。

関連項目：

- [「ターゲットの最適化」 \(ページ 20\)](#)

ソースのボトルネック

パフォーマンスのボトルネックは、Integration Service がソースデータベースからの読み込みを行うときに発生する可能性があります。不適切なクエリ、またはデータベースのネットワークパケットサイズが小さいことによって、ソースのボトルネックが発生する可能性があります。

ソースのボトルネックの特定

セッションログのスレッド統計を読み込んで、ソースがボトルネックになっているかどうかを判断することもできます。Integration Service がトランスフォーメーションまたは writer スレッドよりも reader スレッドで時間を多く使用していれば、それがソースのボトルネックです。

セッションでリレーショナルソースからの読み込みを行う場合、次の方法でソースのボトルネックを特定します。

- Filter トランスフォーメーション
- 読み込みテストマッピング
- データベースクエリー

セッションでフラットファイルソースからの読み込みが行われる場合、ソースにはボトルネックがないと考えられます。

Filter トランスフォーメーションの使用

マッピング内の Filter トランスフォーメーションを使用して、ソースデータの読み込みにかかる時間を測定できます。

各 Source Qualifier の後ろに Filter トランスフォーメーションを追加します。Filter トランスフォーメーションの条件を false に設定すると、データがそのトランスフォーメーションを通過できなくなります。新しいセッションの実行時間が以前と同じ場合は、ソースにボトルネックがあります。

読み込みテストマッピングの使用

読み込みテストマッピングを作成して、ソースのボトルネックを特定できます。読み込みテストマッピングでは、マッピング内のトランスフォーメーションを削除することによって、読み込みクエリーを見つけ出します。

読み込みテストマッピングを作成するには、以下の手順を実行します。

1. 元のマッピングのコピーを作成します。
2. マッピングのコピー内で、ソース、Source Qualifier、およびユーザ作成の任意の結合やクエリーのみを保存します。
3. トランスフォームをすべて削除します。
4. Source Qualifier をファイルターゲットに接続します。

読み込みテストマッピングに対してセッションを実行します。セッションのパフォーマンスが元のセッションのパフォーマンスとあまり変わらない場合、ソースにボトルネックがあると考えられます。

データベースクエリの使用

ソースのボトルネックを特定するには、ソースデータベースに対して読み込みクエリーを直接実行します。

セッションログから、読み込みクエリーを直接コピーします。ISQL などのクエリーツールを使用して、ソースデータベースに対してクエリーを実行します。Windows では、クエリーの結果をファイルにロードすることができます。UNIX では、クエリーの結果を/dev/null にロードすることができます。

クエリーの実行時間およびクエリーが最初のレコードを返すまでにかかる時間を測定します。

ソースのボトルネックの除去

ソースのボトルネックを除去するには、以下のタスクを実行します。

- Integration Service でフラットファイルソースから読み込みを行っている場合は、Integration Service が読み込む行単位のバイト数を設定します。
- データベース管理者に、クエリーを最適化することによってデータベースのパフォーマンスを最適化するように依頼します。
- データベースのネットワークパケットサイズを大きくします。
- インデックスおよびキー制約を設定します。
- データベースクエリーの 2 回の測定間で遅延が大きい場合は、オブティマイザヒントを使用します。

関連項目：

- [「ソースの最適化」 \(ページ 24\)](#)

マッピングのボトルネック

ソースまたはターゲットにボトルネックがないことを確認した場合は、マッピングにボトルネックが存在する可能性があります。

マッピングのボトルネックの特定

マッピングのボトルネックを特定するには、以下のタスクを実行します。

- セッションログからスレッド統計および作業時間統計を読み込みます。Integration Service が writer または reader スレッドよりもトランスフォーメーションスレッドで時間を多く使用していれば、それがトランスフォーメーションのボトルネックです。Integration Service がトランスフォーメーションでより多くの時間を使用していれば、それがトランスフォーメーションスレッド内のボトルネックです。
- パフォーマンスカウンタを分析します。errorrows カウンタおよび rowsinlookupcache カウンタの高い値によって、マッピングのボトルネックが示されます。
- 各ターゲット定義の前に Filter トランスフォーメーションを追加します。データがターゲットテーブルにロードされないように、フィルタ条件を偽に設定します。新しいセッションの実行時間が元のセッションの実行時間と同じ場合は、マッピングにボトルネックがあると考えられます。

マッピングのボトルネックの除去

マッピングのボトルネックを除去するには、マッピングのトランスフォーメーション設定を最適化します。

関連項目：

- [「マッピングの最適化」](#) (ページ 27)

セッションのボトルネック

ソース、ターゲット、またはマッピングにボトルネックがないことを確認できた場合、セッションにボトルネックが存在する可能性があります。キャッシュのサイズやバッファのメモリ容量の不足、およびコミットの間隔が短いことが原因で、セッションのボトルネックが発生することがあります。

セッションのボトルネックの特定

セッションのボトルネックを特定するには、パフォーマンスの詳細を分析します。パフォーマンスの詳細には、入力行、出力行およびエラー行の数など、各トランスフォーメーションに関する情報が表示されます。

セッションのボトルネックの除去

セッションのボトルネックを除去するには、セッションを最適化します。

関連項目：

- [「セッションの最適化」](#) (ページ 43)

システムのボトルネック

ソース、ターゲット、マッピングおよびセッションをチューニングした後、システムのボトルネックを回避するためにシステムのチューニングを検討します。Integration Service ではシステムリソースを使用して、トランスフォーメーションの処理、セッションの実行、およびデータの読み込みと書き込みを行います。また、Integration Service では、システムメモリを使用して、Aggregator、Joiner、Lookup、Sorter、XML および Rank などのトランスフォーメーションのためのキャッシュファイルを作成します。

システムのボトルネックの特定

システムのリソース使用率は、Workflow Monitor で確認できます。システムツールを使用すると、Windows システムおよび UNIX システムをモニタリングできます。

Workflow Monitor を使用したシステムのボトルネックの特定

Workflow Monitor で Integration Service プロパティを表示すると、Integration Service でタスクプロセスを実行しているときのシステムの CPU 使用率、メモリ使用量およびスワップ使用状況を確認できます。パフォーマンスの問題を特定するには、以下の Integration Service プロパティを使用します。

- **CPU%**。CPU 使用率には、システムで実行されている他の外部タスクの使用分が含まれます。
- **メモリ使用量**。メモリ使用率には、システムで実行されている他の外部タスクの使用分が含まれます。メモリ使用量が 95% に近づいた場合、システム上で実行されているタスクが Workflow Monitor で指示された量を使用しているかどうか、またメモリリークが存在するかどうかを確認してください。トラブルシュー

ティングするには、システムツールを使ってセッション実行前後のメモリ使用量を調べ、その結果をセッションを実行中のメモリ使用量と比較します。

- **スワップの使用。**スワップ使用状況は、メモリリークの可能性または同時実行されるタスク数の過多によって発生したページングの実績です。

Windows におけるシステムのボトルネックの特定

システムの情報は、タスク マネージャの[パフォーマンス]タブおよび[プロセス]タブで表示できます。タスク マネージャの [パフォーマンス] タブには、CPU の使用率とメモリの総使用量が表示されています。パフォーマンス モニタを使用すると、より詳細な情報が表示されます。

以下の表に、Windows のパフォーマンスモニタでチャートを作成するために使用できるシステム情報を示します。

プロパティ	説明
パーセントプロセッサ時間	複数の CPU がある場合は、各パーセントごとのプロセッサ時間を監視します。
ページ/秒	ページ/秒が 5 を超えている場合、メモリに過大な負担がかかっている可能性があります（この状態をスラッシングと呼びます）。
物理ディスクパーセント時間	読み取りまたは書き込みの要求を実行するときに、物理ディスクがビジーになるパーセント時間を示します。
物理ディスクキュー長	同一のディスクデバイスへのアクセスを待っているユーザーの数を示します。
秒あたりのサーバー処理バイト合計	サーバーとネットワークの間で送信および受信したバイト数を示します。

UNIX におけるシステムのボトルネックの特定

以下のツールを使用して、UNIX でシステムのボトルネックを特定します。

- top。システム全体のパフォーマンスを表示します。このツールでは、システム、およびシステム上で実行されている個々のプロセスに関する CPU 使用率、メモリ使用量およびスワップ使用状況が表示されます。
- iostat。データベースサーバーに取り付けられているすべてのディスクに対するロード操作を監視します。iostat には、ディスクが物理的にアクティブになっている時間の割合が表示されます。ディスクアレイを使用する場合は、iostat は使用しないで、ディスクアレイに同梱されているユーティリティを使用します。
- vmstat。ディスクのスワップ動作を監視します。
- sar。CPU 使用率、メモリ使用量、およびディスク使用状況に関する詳細システムアクティビティレポートを表示します。このツールは、CPU のロードの監視に使用できます。ユーザ、システム、アイドル時間および待ち時間の使用率を知ることができます。このツールは、ディスクのスワップ動作の監視にも使用できます。

システムのボトルネックの除去

システムのボトルネックを除去するには、以下のタスクを実行します。

- CPU 使用率が 80%を超過した場合は、同時実行タスクの数をチェックしてください。負荷を変更するかまたはグリッドを使って複数ノードにタスクを分散させることを検討する必要があります。負荷が低下しない場合は、プロセッサの追加を検討します。

- スワップが発生する場合は、物理メモリを増設するか、またはディスク上のメモリ集約型アプリケーションの数を減らします。
- メモリが極端に圧迫される場合は（スラッシング）、物理メモリの追加を検討します。
- パーセント時間が高い場合は、PowerCenter のキャッシュをチューニングして、ディスクに書き込む代わりにインメモリキャッシュを使用するようにします。キャッシュをチューニングしても、リクエストがまだキューにあり、ディスクのビジーの割合が最低でも 50%ある場合は、ディスクデバイスを追加するか、速いディスクデバイスにアップグレードしてください。このほか、セッションの各パーティションで異なるディスクを使用することもできます。
- 物理ディスクキュー長が 2 を超える場合は、別のディスクデバイスの追加、または現在のディスクデバイスのアップグレードを検討する必要があります。このほか、reader、writer、およびトランスフォーメーションスレッドでも異なるディスクを使用することができます。
- ネットワーク帯域幅の改善を検討します。
- UNIX システムをチューニングする場合は、主要なデータベースシステムのサーバもチューニングします。
- I/O（%wio）で待機している時間のパーセンテージが高い場合、使用率の低い他のディスクの使用を検討する必要があります。たとえば、ソースデータ、ターゲットデータ、ルックアップ、ランク、および集計キャッシュファイルのすべてが 1 つのディスクに格納されている場合、これらの一部を別のディスクに移すことを検討する必要があります。

関連項目：

- [「ページングの削減」 \(ページ 62\)](#)
- [「システムの最適化」 \(ページ 61\)](#)

第 3 章

ターゲットの最適化

この章では、以下の項目について説明します。

- [フラットファイルターゲットの最適化, 20 ページ](#)
- [インデックスおよびキー制約の削除, 20 ページ](#)
- [データベースのチェックポイント間隔を長く設定, 21 ページ](#)
- [バルクロードの使用, 21 ページ](#)
- [外部ローダーの使用, 21 ページ](#)
- [デッドロックの最小化, 22 ページ](#)
- [データベースのネットワークパケットサイズの拡張, 22 ページ](#)
- [Oracle ターゲットデータベースの最適化, 22 ページ](#)

フラットファイルターゲットの最適化

フラットファイルターゲット用に共有ストレージディレクトリを使用する場合、他のタスクを実行せず、ファイルの保存および管理を専用に行うマシンに共有ストレージディレクトリを作成することで、セッションのパフォーマンスを最適化できます。

Integration Service が単一ノード上で実行されているとき、セッションがフラットファイルターゲットに書き込みを行う場合は、Integration Service プロセスノードに対してローカルなフラットファイルターゲットに書き込むと、セッションのパフォーマンスを最適化できます。

インデックスおよびキー制約の削除

ターゲットテーブルにキー制約またはインデックスを定義している場合は、これらのテーブルへのデータのロードが遅くなります。パフォーマンスを改善するには、インデックスおよびキー制約を削除してからセッションを実行します。インデックスおよびキー制約は、セッションの完了後に再構築することができます。

インデックスおよびキー制約を定期的に削除して再構築する場合は、次の方法によりセッションを実行するたびにその処理を行うことができます。

- ロード前/後のストアードプロシージャを使用します。
- セッション実行前/実行後の SQL コマンドを使用します。

注: パフォーマンスを最適化するには、必要に応じて、制約に基づくロードを実行します。

データベースのチェックポイント間隔を長く設定

Integration Service がデータベースでのチェックポイント実行を待つたびに、パフォーマンスが低下します。チェックポイント数を減らしてパフォーマンスを向上させるには、データベースのチェックポイント間隔を大きくします。

注: チェックポイント数を減らすとパフォーマンスは向上しますが、データベースが予期せずシャットダウンした場合のリカバリ時間も長くなります。

バルクロードの使用

一括ロードの使用により、DB2、Sybase ASE、Oracle、または Microsoft SQL Server のデータベースに大量データを挿入するセッションのパフォーマンスを向上させることができます。一括ロードはセッションのプロパティで設定できます。

一括ロードを行う場合は、Integration Service はデータベースログを無視するため、パフォーマンスが向上します。ただし、データベースログへの書き込みが行われないので、ターゲットデータベースではロールバックを実行できません。その結果、リカバリを実行できない場合があります。一括ロードを実行する場合は、セッションのパフォーマンスを向上させる機能と、不完全なセッションをリカバリする機能のどちらを重視するかを検討する必要があります。

Microsoft SQL Server や Oracle のターゲットに一括ロードを行う場合には、パフォーマンスを向上させるためにコミット間隔を大きく定義します。Microsoft SQL Server と Oracle ではコミットのたびに新しい一括ロードトランザクションが開始されます。コミット間隔を大きくすることで、一括ロードの回数を減らしてパフォーマンスを向上できます。

関連項目：

- [「ターゲットベースのコミット」 \(ページ 47\)](#)

外部ローダーの使用

セッションのパフォーマンスを向上させるには、以下のタイプのターゲットデータベースに外部ローダーを使用するように PowerCenter を設定します。

- IBM DB2 EE または EEE
- Oracle

複数のパーティションを含むパイプラインを使用して Oracle データベースにデータをロードする場合、そのセッションで使用するパーティションと同じ番号で Oracle ターゲットテーブルを作成すると、パフォーマンスが向上します。

- Sybase IQ

UNIX システム上の Integration Service プロセスで Sybase IQ データベースをローカルに使用している場合は、名前付きのパイプからターゲットテーブルにデータを直接ロードすると、パフォーマンスを向上させることができます。グリッドで Integration Service を実行する場合は、リソースをチェックして Sybase IQ をリソースとし、そのリソースをグリッドのすべてのノードから使用できるようにロードバランスを設定します。次に、Workflow Manager で、Sybase IQ リソースを適切なセッションに割り当てます。

- Teradata

デッドロックの最小化

ターゲットに書き込みを試行した際に Integration Service でデッドロックが発生した場合は、デッドロックは同じターゲット接続グループのターゲットにのみ影響します。Integration Service は、他のターゲット接続グループのターゲットにも正常に書き込みを行います。

デッドロックが発生すると、セッションのパフォーマンスが低下することがあります。セッションのパフォーマンスを向上させるために、Integration Service ではセッションでターゲットへの書き込みに使用するターゲット接続グループの数を増やすことができます。セッションで各ターゲットに対して個別のターゲット接続グループを使用するには、各ターゲットインスタンスに対して個別のデータベース接続名を使用します。各接続名に同じ接続情報を指定できます。

データベースのネットワークパケットサイズの拡張

Oracle、Sybase ASE、または Microsoft SQL Server のターゲットに対して書き込みを行う場合、ネットワークパケットのサイズを拡張してパフォーマンスを向上させることができます。ネットワークパケットのサイズを拡張して、大きなデータパケットが一度にネットワークを流れるようにします。書き込み先のデータベースに基づいて、ネットワークパケットのサイズを拡張します。

- **Oracle。** データベースサーバのネットワークパケットのサイズは、listener.ora と tnsnames.ora で拡張できます。必要に応じて、パケットサイズの拡張の詳細について、データベースのマニュアルを参照してください。
- **Sybase ASE および Microsoft SQL Server。** パケットサイズの拡張方法の詳細については、データベースのマニュアルを参照してください。

Sybase ASE または Microsoft SQL Server については、Workflow Manager でリレーショナル接続オブジェクト内のパケットサイズを変更して、データベースサーバのパケットサイズに対応させることも必要です。

Oracle ターゲットデータベースの最適化

ターゲットデータベースに Oracle を使用している場合、ストレージ句、領域の割り当て、およびロールバックまたは取り消しセグメントをチェックすることによって、ターゲットデータベースを最適化できます。

Oracle データベースへの書き込みを行うときに、データベースオブジェクトのストレージ句をチェックします。テーブルで大きな初期値と増分値が使用されていることを確認します。また、データベースでは、テーブルとインデックスデータを別々のテーブルスペース（別々のディスクが望ましい）に格納する必要があります。

Oracle データベースへの書き込みを行うとき、データベースではロード時にロールバックまたは取り消しセグメントを使用します。Oracle のデータベース管理者に依頼して、データベースによってロールバックまたは取り消しセグメントがそれぞれ適切なテーブルスペース（別々のディスクが望ましい）に保存されていることを確認します。また、ロールバックまたは取り消しセグメントには、適切なストレージ句が必要です。

Oracle データベースを最適化するには、Oracle の REDO ログをチューニングします。Oracle データベースでは、REDO ログを使用してロード処理を記録します。REDO ログおよびバッファのサイズが最適であることを確認してください。REDO ログのプロパティは、init.ora ファイルで見ることができます。

Integration Service が単一ノード上で実行されているときに、Oracle インスタンスが Integration Service プロセスノードに対してローカルである場合は、IPC プロトコルを使用して Oracle データベースに接続すると、パフォーマンスを最適化できます。Oracle データベースへの接続は、listener.ora および tnsnames.ora で設定できます。

Oracle データベースの最適化の詳細については、Oracle のマニュアルを参照してください。

第 4 章

ソースの最適化

この章では、以下の項目について説明します。

- [クエリの最適化, 24 ページ](#)
- [条件フィルタの使用, 25 ページ](#)
- [データベースのネットワークパケットサイズの拡張, 25 ページ](#)
- [Oracle データベースソースへの接続, 25 ページ](#)
- [Teradata FastExport の使用, 25 ページ](#)
- [Sybase または Microsoft SQL Server のテーブルを結合するための tempdb の使用, 26 ページ](#)

クエリの最適化

セッションで 1 つの Source Qualifier 内にある複数のソーステーブルを結合する場合、最適化のヒントを利用してクエリーを最適化することによって、パフォーマンスを向上させることができます。また、ORDER BY 句または GROUP BY 句を使用した単独テーブル選択文に対しては、インデックスの追加などの最適化が役立ちます。

通常、データベースオプティマイザでは、ソースデータを処理する際の最も効率的な方法を判定します。ただし、ユーザはデータベースのオプティマイザが認識できないソーステーブルの属性を知っている場合があります。データベース管理者は、ソーステーブルの特定のセットに対するクエリーの実行方法をデータベースに指示するためのオプティマイザヒントを作成できます。

データを読み込むために Integration Service が使用するクエリは、セッションログに記録されます。Source Qualifier トランスフォーマーメーションでも、クエリーを見ることができます。データベース管理者にクエリーの分析を依頼し、オプティマイザヒントやソーステーブルのインデックスを作成します。

クエリーの実行を開始するとき、および PowerCenter でデータの最初のレコードを受け取るときに大きな遅れが発生する場合は、最適化のヒントを使用します。オプティマイザヒントが、すべてのレコードを一度に返すのではなく、できるだけ早くレコードを返すように設定します。これにより、Integration Service は、クエリーの実行と並行してレコードを処理できるようになります。

ORDER BY カラムまたは GROUP BY カラムにインデックスを作成することにより、ORDER BY 句または GROUP BY 句を含むクエリーの効率が向上します。クエリーを最適化したら、SQL オーバーライドオプションを使用して、最適化の際の修正が最大限に活用されるようにします。

また、ソースデータベースで並行クエリーを実行できるように設定して、パフォーマンスを向上させることもできます。平行クエリーの設定の詳細については、データベースのマニュアルを参照してください。

条件フィルタの使用

インデックスが設定されていないことが原因で、ソースデータベース上の簡単なソースフィルタが、パフォーマンスに悪影響を及ぼすことがあります。Source Qualifier で、PowerCenter の条件フィルタを使用すると、パフォーマンスを向上させることができます。

PowerCenter の条件フィルタを使用してパフォーマンスを向上できるかどうかは、セッションによって異なります。たとえば、複数のセッションで同一のソースから同時に読み込みを行う場合、PowerCenter の条件フィルタでパフォーマンスを向上させることがあります。

ただし、ソースデータベース上のソースデータにフィルタをかける方が、実行速度が速くなるセッションもあります。データベースフィルタおよび PowerCenter フィルタの両方でセッションをテストして、どの方法でパフォーマンスを向上させるべきかを判定できます。

データベースのネットワークパケットサイズの拡張

Oracle、Sybase ASE、または Microsoft SQL Server のソースから読み込む場合、ネットワークパケットのサイズを拡張してパフォーマンスを向上させることができます。ネットワークパケットのサイズを拡張して、大きなデータパケットが一度にネットワークを流れるようにします。読み込み対象となる以下のデータベースの種類に応じて、ネットワークパケットのサイズを拡張します。

- **Oracle。** データベースサーバのネットワークパケットのサイズは、listener.ora と tnsnames.ora で拡張できます。必要に応じて、パケットサイズの拡張の詳細について、データベースのマニュアルを参照してください。
- **Sybase ASE および Microsoft SQL Server。** パケットサイズの拡張方法の詳細については、データベースのマニュアルを参照してください。

Sybase ASE または Microsoft SQL Server については、Workflow Manager でリレーショナル接続オブジェクト内のパケットサイズを変更して、データベースサーバのパケットサイズに対応させることも必要です。

Oracle データベースソースへの接続

Integration Service が単一ノード上で実行されているときに、Oracle インスタンスが Integration Service プロセスノードに対してローカルである場合は、IPC プロトコルを使用して Oracle データベースに接続することによって、パフォーマンスを最適化できます。Oracle データベースへの接続は、listener.ora および tnsnames.ora で設定できます。

Teradata FastExport の使用

FastExport は複数の Teradata セッションを使用して、Teradata データベースから大容量のデータを素早くエクスポートするツールです。FastExport を使用して Teradata ソースをすばやく読み込む PowerCenter セッションを作成できます。FastExport を使用するには、Teradata ソースデータベースとのマッピングを作成します。そのセッション中は、Relational reader に代わって FastExport reader を使用します。セッションにエクスポートしたい Teradata テーブルで FastExport 接続を使用します。

Sybase または Microsoft SQL Server のテーブルを結合するための tempdb の使用

Sybase または Microsoft SQL Server のデータベースで大きいテーブルを結合すると、メモリを効率的に割り当てるためのインメモリデータベースとして tempdb を作成することでパフォーマンスが向上する可能性があります。詳細については、Sybase または Microsoft SQL Server のマニュアルを参照してください。

第 5 章

マッピングの最適化

この章では、以下の項目について説明します。

- [マッピングの最適化の概要, 27 ページ](#)
- [フラットファイルソースの最適化, 27 ページ](#)
- [Single-Pass 読み込みの設定, 28 ページ](#)
- [パススルーマッピングの最適化, 29 ページ](#)
- [フィルタの最適化, 29 ページ](#)
- [データタイプ変換の最適化, 30 ページ](#)
- [式の最適化, 30 ページ](#)
- [External Procedure の最適化, 32 ページ](#)

マッピングの最適化の概要

マッピングレベルの最適化はその実装に時間がかかりますが、セッションのパフォーマンスを大幅に向上させることができます。マッピングレベルの最適化は、ターゲットおよびソースの最適化が完了した後にのみ実施できます。

通常、マッピングの最適化では、マッピング内のトランスフォーメーションの数を減らし、トランスフォーメーション間の不要なリンクを削除します。最大限の処理を行うには、トランスフォーメーションと式をできるだけ少なくしてマッピングを設定します。トランスフォーメーション間の不要なリンクを削除して、移動するデータの量を最小限に抑えます。

フラットファイルソースの最適化

フラットファイルソースを最適化するには、次のタスクを実行します。

- 連続行のバッファ長を最適化します。
- 区切りフラットファイルソースを最適化します。
- XML およびフラットファイルソースの最適化

連続行のバッファ長の最適化

セッションでフラットファイルソースからの読み込みを行う場合は、Integration Service で各行の読み込みバイト数を設定することにより、セッションのパフォーマンスを向上させることができます。デフォルトでは、Integration Service は 1 行につき 1024 バイトを読み込みます。ソースファイル内の各行のバイト数がデフォルト設定の値より少ない場合、セッションのプロパティで連続行のバッファ長の値を低くすることができます。

区切りフラットファイルソースの最適化

ソースが区切りフラットファイルの場合、区切り文字を指定してソースファイルのデータの列を分ける必要があります。また、エスケープ文字も指定する必要があります。区切り文字の前にエスケープ文字を入れると、Integration Service は区切り文字を通常の文字として読み込みます。ソースファイルが引用符やエスケープ文字を含まないようにすれば、セッションのパフォーマンスを向上できます。

XML およびフラットファイルソースの最適化

XML ファイルにはタグ情報があるので、通常はフラットファイルよりもサイズが大きくなります。XML ファイルの大きさは、XML ファイルのタグ付けのレベルによって異なります。タグの数が多いと、ファイルサイズは大きくなります。その結果、Integration Service では XML ソースを読み込みキャッシュするのに時間がかかることもあります。

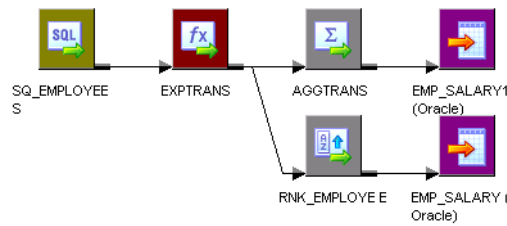
Single-Pass 読み込みの設定

Single-Pass 読み込みでは、1 つの Source Qualifier を使用して複数のターゲットに値を入れることができます。同じソースを使用する複数のセッションがある場合は、Single-Pass 読み込みの使用を検討します。各マッピングのトランスフォーメーションロジックを 1 つのマッピングに結合し、各ソースで 1 つの Source Qualifier を使用することができます。統合サービスは、各ソースを一度読み込んでからデータを異なるパイプラインに送信します。状況に応じて、パイプラインの組み合わせや、パイプライン以外の組み合わせにより、すべてのパイプラインで特定の行を使用することもできます。

たとえば、Purchasing ソーステーブルがあり、そのソースを毎日使用して集計およびランク付けを行うとします。Aggregator トランスフォーメーションと Rank トランスフォーメーションを別個のマッピングおよびセッションに分けた場合、統合サービスに同じソーステーブルを 2 回読み込ませることになります。ただし、1 つの Source Qualifier を有する 1 つのマッピングに集計およびランキングロジックを含めると、統合サービスでは、一度 Purchasing ソーステーブルを読み込んでから、異なるパイプラインに適切なデータを送信します。

Single-Pass 読み込みを利用するようにマッピングを変更する場合は、マッピングに共通する処理を抽出することによって、この機能を最適化します。たとえば、Aggregator と Rank の両方のトランスフォーメーションにおいて Price ポートからパーセンテージを減算する必要がある場合は、パイプラインを分割する前にパーセンテージの減算処理を行うことによって処理を最小限に抑えることができます。Expression トランスフォーメーションを使用してパーセンテージを減算してから、トランスフォーメーション実行後にマッピングを分割します。

以下の図に、Expression トランスフォーメーションの後にマッピングが分かれる Single-Pass 読み込みを示します。



パススルーマッピングの最適化

パススルーマッピングのパフォーマンスを最適化できます。データを、他のトランスフォーメーションを使用せずに直接ソースからターゲットに渡す場合は、Source Qualifier トランスフォーメーションを直接ターゲットに接続します。基本操作ウィザードを使用してパススルーマッピングを作成する場合は、このウィザードによって、Source Qualifier トランスフォーメーションとターゲットの間に Expression トランスフォーメーションが作成されます。

フィルタの最適化

データをフィルタするには、以下のトランスフォーメーションのいずれかを使用します。

- **Source Qualifier トランスフォーメーション。** Source Qualifier トランスフォーメーションによって、リレーショナルソースの行がフィルタリングされます。
- **Filter トランスフォーメーション。** Filter トランスフォーメーションはマッピング内のデータをフィルタリングします。Filter トランスフォーメーションは、すべての種類のソースからの行をフィルタリングします。

マッピングからのレコードにフィルタを適用する場合、データフローの早めの段階でフィルタリングを行うことによって効率を上げることができます。Source Qualifier トランスフォーメーションのフィルタを使用して、ソースから行を削除します。Source Qualifier トランスフォーメーションは、リレーショナルソースから抽出した行セットを制限します。

Source Qualifier トランスフォーメーションのフィルタを使用できない場合は、Filter トランスフォーメーションを使用して、できるだけ Source Qualifier トランスフォーメーションの近くまで移動してデータフローの初期段階で不必要なデータを削除します。Filter トランスフォーメーションは、ターゲットに送信される行セットを制限します。

フィルタ条件では、複雑な式を使用しないようにします。Filter トランスフォーメーションを最適化するには、フィルタ条件で簡単な整数式または真偽式を使用します。

注: 拒否された行を保持する必要がある場合、Filter または Router トランスフォーメーションを使用して、拒否された行を Update Strategy トランスフォーメーションから削除します。

データタイプ変換の最適化

不要なデータ型変換を除去することにより、パフォーマンスが向上します。たとえば、Integer カラムから Decimal カラムへデータを移動した後に再び Integer カラムにそのデータを戻すようなマッピングの場合、不必要なデータ型を変換することによってパフォーマンスが低下します。可能な限り、不必要なデータ型の変換をマッピングから取り除きます。

システムのパフォーマンスを向上させるため、次のデータ型の変換を実行します。

- **Lookup トランスフォーメーションと Filter トランスフォーメーションを使用して比較を実行する場合には、他のデータ型の代わりに整数値を使用する。**たとえば、多くのデータベースでは、米国の郵便番号情報が Char データ型または Varchar データ型として格納されます。郵便番号のデータを Integer データ型に変換した場合、ルックアップデータベースでは郵便番号の 94303-1234 を 943031234 という値で格納します。これで、郵便番号に基づいたルックアップの比較条件の処理が高速化します。
- **ポート間変換を通じて、ソースデータを文字列に変換して、セッションのパフォーマンスを向上させる。**ターゲット内のポートは文字列のままにしておくことも、Date/Time ポートに変更することもできます。

式の最適化

トランスフォーメーションで使用する式を最適化することもできます。可能な場合は、パフォーマンスを低下させている式を見つけて簡素化します。

パフォーマンスを低下させている式を見つけるため、次のタスクを実行します。

1. マッピングから 1 つずつ式を削除します。
 2. マッピングを実行して、トランスフォーメーション抜きでマッピングを実行したときの時間を測定します。
- セッションの実行時間が著しく異なる場合は、パフォーマンスを低下させている式を最適化する方法を探します。

共通ロジックの抽出

マッピングによって複数の箇所で同じ処理が実行される場合、その処理をマッピングの初期段階に移すことによって、マッピングによる処理の実行回数を減らします。たとえば、マッピングに 5 つのターゲットテーブルがある場合を検討しましょう。各ターゲットには、社会保障番号のルックアップが必要です。この場合、ルックアップを 5 回実行する代わりに、データフローが分かれる前のマッピングに Lookup トランスフォーメーションを配置します。次に、ルックアップの結果を 5 つのターゲットすべてに渡します。

集計関数呼び出しの最小化

式を書き込む場合、集計関数の呼び出しをできる限り減らします。集計関数が呼び出されるたびに、Integration Service ではデータの検索とグループ化を行う必要が生じます。たとえば、次の式では、Integration Service は最初に COLUMN_A を読み込んでその合計を求めます。次に、COLUMN_B を読み込んでその合計を求め、最後に 2 つの合計の和を求めます。

```
SUM(COLUMN_A) + SUM(COLUMN_B)
```

次の例のように集計関数の呼び出しを減らした場合、Integration Service は COLUMN_A を COLUMN_B に足した後でその合計を求めます。

```
SUM(COLUMN_A + COLUMN_B)
```

ローカル変数での共通の式の置き換え

1つのトランスフォーメーションにおいて同じ式が複数回使用されている場合は、その部分式をローカル変数にすることができます。ローカル変数はトランスフォーメーション内でのみ使用できます。ただし、変数の計算は一度で済むので、パフォーマンスを向上させることができます。

文字列演算の代わりに数値演算を実行

Integration Service では、数値演算の方が文字列演算よりも高速に処理されます。たとえば、2つの列（EMPLOYEE_NAME と EMPLOYEE_ID）で大量のデータを検索する場合、EMPLOYEE_ID にルックアップを設定することでパフォーマンスが改善されます。

Char 対 Char および Char 対 Varchar の比較の最適化

Integration Service が CHAR 列と VARCHAR 列の比較を行う場合、行の末尾に空白が検出されるたびに処理速度が低下します。Informatica Administrator で Integration Service を設定する場合に、TreatCHARasCHARonRead オプションを使用すると、Integration Service は Char 型ソースフィールドの末尾の空白を削除しません。

LOOKUP の代わりに DECODE を選択

LOOKUP 関数を使用すると、Integration Service ではデータベース内のテーブルを検索する必要が生じます。DECODE 関数を使用するとルックアップ値が式自体に取り込まれ、Integration Service が個々のテーブルを検索する必要がなくなります。したがって、検索する値が変化せず、その数も少ない場合は、DECODE を使用するとパフォーマンスが向上します。

関数の代わりに演算子を使用

Integration Service では、関数を使用して書かれた式よりも、演算子を使用して書かれた式の方がより高速で読み込まれます。したがって、できるだけ演算子を使用して式を記述してください。たとえば、CONCAT 関数をネストした次のような式があるとして。

```
CONCAT( CONCAT( CUSTOMERS.FIRST_NAME, ' ' ) CUSTOMERS.LAST_NAME)
```

||演算子を使用すると、上記の式を次のように表すことができます。

```
CUSTOMERS.FIRST_NAME || ' ' || CUSTOMERS.LAST_NAME
```

IIF 関数の最適化

IIF 関数では、値およびアクションを返すことができるため、式をより簡潔にできます。たとえば、FLG_A、FLG_B、FLG_C という 3 つの Y/N フラグを含むソースがあり、各フラグの値に基づく値を返したいとします。

次の式を使用します。

```
IIF( FLG_A = 'Y' and FLG_B = 'Y' AND FLG_C = 'Y',
```

```
    VAL_A + VAL_B + VAL_C,
```

```
    IIF( FLG_A = 'Y' and FLG_B = 'Y' AND FLG_C = 'N',
```

```
        VAL_A + VAL_B ,
```

```
        IIF( FLG_A = 'Y' and FLG_B = 'N' AND FLG_C = 'Y',
```

```
            VAL_A + VAL_C,
```

```
            IIF( FLG_A = 'Y' and FLG_B = 'N' AND FLG_C = 'N',
```

```

VAL_A ,
IIF( FLG_A = 'N' and FLG_B = 'Y' AND FLG_C = 'Y',
VAL_B + VAL_C,
IIF( FLG_A = 'N' and FLG_B = 'Y' AND FLG_C = 'N',
VAL_B ,
IIF( FLG_A = 'N' and FLG_B = 'N' AND FLG_C = 'Y',
VAL_C,
IIF( FLG_A = 'N' and FLG_B = 'N' AND FLG_C = 'N',
0.0,
))))))

```

この式では、8 個の IIF、16 個の AND、および少なくとも 24 回の比較が必要です。

IIF 関数を利用すると、上記の式を次のように表すことができます。

```
IIF(FLG_A='Y', VAL_A, 0.0)+ IIF(FLG_B='Y', VAL_B, 0.0)+ IIF(FLG_C='Y', VAL_C, 0.0)
```

この式で使用する IIF は 3 つ、比較は 2 つ、加算は 2 つとなり、その結果、セッションが高速化されます。

式の評価

どの式がパフォーマンスを低下させているか不明な場合は、式のパフォーマンスを評価して問題を見つけ出します。

式のパフォーマンスを評価する次のステップを実行します。

1. 元の式を使用してセッションを計測します。
2. マッピングをコピーし、複雑な式の半分を定数に置き換えます。
3. 編集したセッションを実行して時間を計測します。
4. マッピングのコピーをもう 1 つ作成し、複雑な式の残りの半分を定数で置き換えます。
5. 編集したセッションを実行して時間を計測します。

External Procedure の最適化

External Procedure で入力グループからの読み込みを切り換える必要がある場合は、入力データをブロックする必要があります。ブロック機能を使わない場合、入力データをバッファリングする手続きコードを書く必要があります。通常はセッションのパフォーマンスを低下させるバッファリングの代わりに、入力データをブロックすることができます。

たとえば、2 つの入力グループで External procedure を作成するとします。この External Procedure は、最初の入力グループのデータを 1 行読み込んでから、2 つ目の入力グループのデータを 1 行読み込みます。ブロックを使用すれば、一方の入力グループからのデータを処理している間は他の入力グループからのデータフローをブロックする External Procedure コードを書くことができます。データをブロックする External Procedure コードを書くと、そのプロセスではソースデータをバッファにコピーする必要がないため、パフォーマンスが向上します。しかし、データを処理する準備ができるまで、バッファを割り当てて入力グループからバッファにデータをコピーする External Procedure を書くこともできます。ソースデータのバッファへのコピーはパフォーマンスを低下させます。

第 6 章

トランスフォーメーションの最適化

この章では、以下の項目について説明します。

- [Aggregator トランスフォーメーションの最適化, 33 ページ](#)
- [Custom トランスフォーメーションの最適化, 35 ページ](#)
- [Joiner トランスフォーメーションの最適化, 35 ページ](#)
- [Lookup トランスフォーメーションの最適化, 36 ページ](#)
- [ノーマライズトランスフォーメーションの最適化, 39 ページ](#)
- [Sequence Generator トランスフォーメーションの最適化, 39 ページ](#)
- [Sorter トランスフォーメーションの最適化, 40 ページ](#)
- [Source Qualifier トランスフォーメーションの最適化, 41 ページ](#)
- [SQL トランスフォーメーションの最適化, 41 ページ](#)
- [XML トランスフォーメーションの最適化, 41 ページ](#)
- [トランスフォーメーションエラーの除去, 42 ページ](#)

Aggregator トランスフォーメーションの最適化

Aggregator トランスフォーメーションではデータを処理する前にそれをグループ化しなければならないため、パフォーマンスが低下することがよくあります。Aggregator トランスフォーメーションは、中間のグループ結果を格納するための追加メモリを必要とします。

Aggregator トランスフォーメーションのパフォーマンスを最適化するには、以下のガイドラインに従います。

- 単純なカラム別にグループ化する。
- ソート済み入力を使用する。
- 差分集計を使用する。
- 集計する前にデータをフィルタリングする。
- ポート接続数を制限する。

単純なカラム別のグループ化

単純なカラム別のグループ化を行う際は、Aggregator トランスフォーメーションを最適化することができません。可能な場合には、GROUP BY で使用するカラムでは、文字列や日付ではなく数値を使用します。アグリゲータの式が複雑にならないようにします。

ソート済み入力の使用

セッションのパフォーマンスを向上させるには、Aggregator トランスフォーメーションのデータをソートします。データのソートには、Sorted Input オプションを使用します。

Sorted Input オプションは、集計キャッシュの使用頻度を低くします。Sorted Input オプションを使用すると、Integration Service はすべてのデータがグループ別にソートされているものと想定します。Integration Service は 1 つのグループに対する行を読み込むときに、読み込みながら集計計算を実行します。必要に応じて、メモリにグループ情報を格納します。

Sorted Input オプションは、キャッシュに格納されるデータ量をセッション中に減らして、パフォーマンスを向上させます。このオプションを Source Qualifier の Number of Sorted Ports オプションまたは Sorter トランスフォーメーションと組み合わせて使用すると、ソート済みデータを Aggregator トランスフォーメーションに渡すことができます。

複数のパーティションが含まれるセッションで Sorted Input オプションを使用すると、パフォーマンスが向上します。

差分集計の使用

ターゲットの半分未満のみに影響するソースから変更内容を取得できれば、差分集計を使用して Aggregator トランスフォーメーションのパフォーマンスを最適化できます。

差分集計を使用すると、取得したソースの変更がセッションで集計計算に適用されます。Integration Service では、ソース全体を処理しないでターゲットの差分のみを更新し、セッションを実行するたびに同じ計算を再計算します。

ディスクにページングせずにすべてのデータをメモリ内に格納できるように、インデックスキャッシュサイズおよびデータキャッシュサイズを増やすことができます。

関連項目：

- [「キャッシュサイズの拡張」 \(ページ 47\)](#)

集計前のデータのフィルタリング

集計する前にデータをフィルタリングします。マッピングで Filter トランスフォーメーションを使用する場合、そのあとに Aggregator トランスフォーメーションを置いて、不要な集計を減らしてください。

ポート接続の制限

接続される入出力ポートまたは出力ポートの数を制限することにより、Aggregator トランスフォーメーションがデータキャッシュに格納するデータ量を減らします。

Custom トランスフォーメーションの最適化

Integration Service では Custom トランスフォーメーションプロシージャに対して 1 行ずつ渡したり、配列内の行をブロック単位で渡したりすることができます。手続きコードを作成して、手続きが 1 行ずつ受け取るか、ブロック単位で受け取るかを指定できます。

ブロック単位で手続きが行を受け取ると、パフォーマンスが向上します。

- Integration Service やプロシージャが行う関数呼び出しの数を減らすことができます。Integration Service による入力行通知関数の呼び出し回数が減り、プロシージャによる出力通知関数の呼び出し回数が減ります。
- データに対するメモリアクセススペースの局所性を高めることができます。
- 手続きコードを作成し、データの各行ではなく、ブロック単位にアルゴリズムを実行するようにできます。

Joiner トランスフォーメーションの最適化

Joiner トランスフォーメーションでは、中間結果を格納するための追加領域を実行時に必要とするため、パフォーマンスが低下することがあります。Joiner トランスフォーメーションを最適化する必要があるかどうかを判断するには、Joiner パフォーマンスカウンタの情報を表示します。

次のヒントを使用して、Joiner トランスフォーメーションによるセッションのパフォーマンスを向上できます。

- **重複キー値が少ない方のソースをマスターとして指定する。** Integration Service は、ソート済み Joiner トランスフォーメーションを処理するときに、一度に 100 個の一意のキーについての行をキャッシュに格納します。マスターソースに同じキー値を持つ多数の行が含まれる場合、Integration Service はより多くの行をキャッシュに格納する必要があるため、パフォーマンスが低下することがあります。
- **行が少ない方のソースをマスターとして指定する。** セッションの間、Joiner トランスフォーメーションは明細ソースの各行をマスターソースと比較します。マスター内の行が少なければ、結合のための比較が繰り返される回数も少なくなり、その結果、結合プロセスが高速になります。
- **可能な場合は、データベース内で結合を実行する。** データベース内で結合を実行すると、セッション内で結合を実行するよりも高速で行えます。パフォーマンスは、使用するデータベース結合の種類によっても変わってきます。ノーマルジョインは、アウタージョインよりも高速で、結果的にレコード数が少なくて済みます。場合によって、たとえば 2 つの異なるデータベースまたはフラットファイルシステムとテーブルを結合する場合は、これが不可能なこともあります。

データベース内で結合を実行したい場合は、次のオプションを使用します。

- セッション実行前に起動されるストアドプロシージャを作成して、データベース内でテーブルを結合する。
- Source Qualifier トランスフォーメーションを使用して結合を実行する。
- **可能な場合は、ソート済みデータを結合する。** セッションのパフォーマンスを改善するには、ソート済みの入力を使用するように Joiner トランスフォーメーションを設定します。Joiner トランスフォーメーションでソート済みデータを使うように設定すると、Integration Service はディスクの入出力を最小化してパフォーマンスを高めます。大量のデータセットを扱う場合に、パフォーマンスを最大限に向上させることができます。未ソート Joiner トランスフォーメーションの場合、行の比較的少ないソースをマスターソースとして指定します。

Lookup トランスフォーメーションの最適化

Lookup テーブルがマッピングにおいてソーステーブルと同じデータベースにあり、キャッシュが使用できない場合は、Lookup トランスフォーメーションを使用する代わりに、ソースデータベース内でテーブル同士を結合します。

Lookup トランスフォーメーションを使用する場合、次のタスクを実行してパフォーマンスを向上させることができます。

- 最適なデータベースドライバを使用する。
- Lookup テーブルをキャッシュする。
- Lookup 条件を最適化する。
- Lookup の行をフィルタする。
- Lookup テーブルをインデックスに入れる。
- 複数の Lookup を最適化する。
- パイプライン Lookup トランスフォーメーションを作成して、ルックアップソースを作成するパイプラインでパーティションを設定する。

最適なデータベースドライバの使用

Integration Service は、ネイティブデータベースドライバまたは ODBC ドライバを使用して Lookup テーブルに接続することができます。ネイティブデータベースドライバは、ODBC ドライバよりも高いパフォーマンスを得ることができます。

ルックアップテーブルのキャッシュ

マッピングに Lookup トランスフォーメーションが含まれている場合は、ルックアップキャッシングを有効にします。キャッシングを有効にしている場合、Integration Service はルックアップテーブルをキャッシュに格納して、セッション中にルックアップキャッシュを参照します。このオプションを有効にしていない場合、Integration Service は行ごとにルックアップテーブルを参照します。

Lookup テーブルをキャッシュに格納するかどうかに関わらず、Lookup クエリーの結果および処理は同じです。ただし、ルックアップキャッシュを使用するとセッションのパフォーマンスを向上させることができます。通常、必要なサイズが 300 MB 未満の Lookup テーブルは、キャッシュに入れます。

Lookup トランスフォーメーションのパフォーマンスをさらに改善するには、以下のタスクを実行します。

- 適切なキャッシュタイプを使用する。
- コンカレントキャッシュを有効にする。
- Lookup 条件の一致を最適化する。
- キャッシュに入れるレコード数を減らす。
- ORDER BY 文を上書きする。
- メモリの多いマシンを使用する。

関連項目：

- [「キャッシュ」 \(ページ 46\)](#)

キャッシュのタイプ

次のキャッシュのタイプを使用して、パフォーマンスを向上させることができます。

- **共有キャッシュ。**複数のトランスフォーメーション間でルックアップキャッシュを共有できます。名前なしキャッシュを同じマッピング内のトランスフォーメーション間で共有することができます。名前付きキャッシュを同じマッピング内、または異なるマッピング内のトランスフォーメーション間で共有することができます。
- **永続キャッシュ。**キャッシュファイルを保存して再使用するには、永続キャッシュを使用できるようにトランスフォーメーションを設定します。この機能は、セッションの実行間でルックアップテーブルが変更されていないことが明らかな場合に使用します。Integration Service はメモリキャッシュをデータベースではなくキャッシュファイルから作成するので、永続キャッシュを使用するとパフォーマンスが向上します。

コンカレントキャッシュの有効化

Lookup トランスフォーメーションを含む Integration Service がセッションを処理する場合、Integration Service はキャッシュを使用する Lookup トランスフォーメーションでデータの最初の行を処理するときに、メモリにキャッシュを構築します。マッピングに複数の Lookup トランスフォーメーションがある場合、Lookup トランスフォーメーションによってデータの最初の行が処理されると、Integration Service は順番にキャッシュを作成します。これにより、Lookup トランスフォーメーションの処理は遅くなります。

コンカレントキャッシュを有効にすると、パフォーマンスを向上させることができます。並行処理する追加パイプライン数が 1 つまたは複数に設定されると、Integration Service は順番ではなく同時にキャッシュを作成します。Aggregator、Joiner、または Sorter トランスフォーメーションなど、処理するのに時間のかかるアクティブなトランスフォーメーションがセッションに多数存在していると、パフォーマンスを最大限に向上させることができます。複数のコンカレントパイプラインを有効にすると、Integration Service はキャッシュを構築するまでアクティブセッションを待たなくなります。パイプライン内の他の Lookup トランスフォーメーションも、同時にキャッシュを作成します。

ルックアップ条件の一致の最適化

Lookup トランスフォーメーションでルックアップキャッシュデータと Lookup 条件が一致すると、最初に一致した値と最後に一致した値を判断するため、データをソートし並べ替えます。Lookup 条件に一致する値を何か返すように、トランスフォーメーションを設定できます。Lookup トランスフォーメーションが一致する値を何か返すよう設定すると、トランスフォーメーションは Lookup 条件に一致する最初の値を返します。一致する最初の値を返すか、または最後の値を返すようにトランスフォーメーションを設定すると、通常どおりにすべてのポートをインデックスに入れません。一致する値を使用すると、パフォーマンス低下の原因でもある、トランスフォーメーションによるすべてのポートのインデックスを行わないため、パフォーマンスを改善することができます。

キャッシュされる行数の削減

キャッシュに入れるレコード数を減らして、パフォーマンスを向上させることができます。Lookup SQL オーバーライドオプションを使用すると、デフォルトの SQL 文に WHERE 句を追加できます。

ORDER BY 文の上書き

デフォルトでは、Integration Service はキャッシュされた Lookup に対して ORDER BY 文を生成します。ORDER BY 文にはすべての Lookup ポートが含まれます。パフォーマンスを向上させるには、デフォルトの ORDER BY 文を使用しないで、カラム数の少ない ORDER BY を上書きで入力します。

上書きで ORDER BY 文を入力しても、Integration Service は常に ORDER BY 文を生成します。ORDER BY 上書きのあとに 2 個のダッシュ (--) を挿入して、生成した ORDER BY 文を抑止します。

たとえば、Lookup トランスフォーメーションが次の Lookup 条件を使用するとします。

```
ITEM_ID = IN_ITEM_ID  
PRICE <= IN_PRICE
```

Lookup トランスフォーメーションには、マッピングで使用される 3 つの Lookup ポート (ITEM_ID、ITEM_NAME、および PRICE) が含まれます。ORDER BY 文を入力した場合、Lookup 条件にポートを入力したのと同じ順番でカラムを入力します。また、すべてのデータベース予約語は引用符で囲みます。

Lookup SQL オーバーライドで次の Lookup クエリーを入力します。

```
SELECT ITEMS_DIM.ITEM_NAME, ITEMS_DIM.PRICE, ITEMS_DIM.ITEM_ID FROM ITEMS_DIM ORDER BY ITEMS_DIM.ITEM_ID,  
ITEMS_DIM.PRICE --
```

メモリの多いマシンの使用

セッションのパフォーマンスを向上させるには、大容量のメモリを搭載した Integration Service ノード上でセッションを実行します。マシンに負荷をかけない範囲で、インデックスキャッシュサイズおよびデータキャッシュサイズをできる限り増やします。Integration Service ノードに十分なメモリがある場合は、ディスクにページングしなくてもすべてのデータをメモリ内に保存できるよう、キャッシュを増やします。

ルックアップ条件の最適化

複数のルックアップ条件を入れる場合は、ルックアップのパフォーマンスを最適化するために、次の順序で条件を指定します。

- 等しい (=)
- より小さい (<)、より大きい (>)、より小さいまたは等しい (<=)、より大きいまたは等しい (>=)
- 等しくない (!=)

Lookup 行のフィルタリング

ルックアップキャッシュ作成時にソースから検索される Lookup 行の数を減らすには、フィルタ条件を作成します。

ルックアップテーブルのインデックス処理

Integration Service は、ルックアップ条件カラムの値に対してクエリ、ソート、および比較を行う必要があります。インデックスには、Lookup 条件で使用する各カラムを含めなければなりません。

次の種類の Lookup において、パフォーマンスを向上させることができます。

- **キャッシュを使用したルックアップ。**パフォーマンスを向上させるには、ルックアップ ORDER BY 文のカラムにインデックスを付けます。セッションログには ORDER BY 文が含まれます。
- **キャッシュを使用しないルックアップ。**パフォーマンスを向上させるには、ルックアップ条件のカラムにインデックスを付けます。Integration Service は、Lookup トランスフォーメーションに入る各行に対して SELECT 文を発行します。

複数のルックアップの最適化

マッピングに複数のルックアップが含まれている場合、キャッシングが有効になっていてヒープメモリが十分にあっても、ルックアップによりパフォーマンスが低下することがあります。一番多くのデータを問い合わせる Lookup トランスフォーメーションを探して、全体のパフォーマンスを向上させることができます。

どの Lookup トランスフォーメーションが多くのデータを処理するかを判断するには、それぞれの Lookup トランスフォーメーションの `Lookup_rowsinlookupcache` カウンタを調べます。このカウンタの値が大きい Lookup トランスフォーメーションがあれば、その Lookup 式をチューニングすることにより効果を得られます。このような式を最適化すると、セッションのパフォーマンスが向上します。

関連項目：

- [「式の最適化」 \(ページ 30\)](#)

パイプライン Lookup トランスフォーメーションの作成

パイプライン Lookup トランスフォーメーションが含まれるマッピングには、部分的なパイプラインが含まれます。このパイプラインには、ルックアップソースおよび Source Qualifier があります。Integration Service では、このパイプラインでルックアップソースのデータが処理されます。Lookup トランスフォーメーションが含まれるパイプラインにルックアップソースのデータが渡され、キャッシュが作成されます。

部分的なパイプラインは、セッションプロパティ内の個別のターゲットのロード順グループです。このパイプラインに複数のパーティションを設定すると、パフォーマンスを改善できます。

ノーマライザトランスフォーメーションの最適化

ノーマライザトランスフォーメーションでは行が生成されます。パフォーマンスを最適化するには、ターゲットのできる限り近くにノーマライザトランスフォーメーションを配置します。

Sequence Generator トランスフォーメーションの最適化

Sequence Generator トランスフォーメーションを最適化するには、再利用可能な Sequence Generator を作成して、複数のマッピング内で同時に使用します。また、キャッシュされる値の数プロパティも設定します。

キャッシュされる値の数プロパティでは、Integration Service によって一度にキャッシュされる値の数を指定します。Number of Cached Value の数値は、小さすぎる値にしないでください。Number of Cached Values は、1,000 より大きい値に設定する必要があります。

キャッシュ値がない場合は、Number of Cache Values に 0 を設定します。キャッシュを使用しない Sequence Generator トランスフォーメーションは、キャッシュを必要とするものよりも高速です。

シーケンスジェネレータトランスフォーメーションで CURRVAL ポートを接続すると、Integration Service は各ブロックで 1 行ずつ処理します。マッピングで NEXTVAL ポートのみを接続すると、パフォーマンスを最適化できます。

関連項目：

- [「Sequence Generator トランスフォーメーションの最適化」 \(ページ 55\)](#)

Sorter トランスフォーメーションの最適化

次のタスクを実行して、Sorter トランスフォーメーションを最適化します。

- データをソートするのに十分なメモリを割り当てる。
- Sorter トランスフォーメーション内にある各パーティションに異なるワークディレクトリを指定する。
- PowerCenter Integration Service を ASCII モードで実行します。

メモリの割り当て

最適なパフォーマンスを実現するには、Integration Service ノードで利用可能な物理 RAM 量以下の値を Sorter キャッシュサイズに設定します。Sorter トランスフォーメーションを使用してデータをソートするには、最低 16MB の物理メモリを割り当てます。デフォルトでは、Sorter キャッシュサイズは 16,777,216 バイトに設定されます。Integration Service がデータをソートするのに十分なメモリを割り当てられない場合、セッションは失敗します。

入力されるデータの量が Sorter キャッシュサイズよりも大きい場合、Integration Service はデータを一時的に Sorter トランスフォーメーションのワークディレクトリに保存します。Integration Service がデータをワークディレクトリに保存する場合、最低でも入力されるデータの量の 2 倍のディスク領域が必要です。入力されるデータの量が Sorter キャッシュサイズよりはるかに大きい場合、Integration Service はワークディレクトリで利用可能なディスク領域の 2 倍の量よりさらに多くの量を必要とする可能性があります。

注: セッションログには、ソータトランスフォーメーションに対する入力行数および入力データのサイズが含まれます。

たとえば、Integration Service がソータトランスフォーメーションを処理するときには以下のメッセージが表示されます。

```
SORT_40422 End of output from Sorter Transformation [srt_1_Billion_FF]. Processed 999999999 rows (866325637228 input bytes; 868929593344 temp I/O bytes)
```

このメッセージでは、「999999999」が入力行数で、「866325637228」が入力行のサイズです。

パーティションのワークディレクトリ

Integration Service はデータをソートするとき、一時ファイルを作成します。一時ファイルは、ワークディレクトリに保存されます。Integration Service ノードの任意のディレクトリをワークディレクトリとして使用できるように指定できます。デフォルトでは、Integration Service は \$PMTempDir サービスプロセス変数に指定されている値を使用します。

Sorter トランスフォーメーションを使ってセッションをパーティション化する場合、パイプライン内のパーティションごとに異なるワークディレクトリを指定できます。セッションのパフォーマンスを向上させるためには、Integration Service のノード上の物理的に異なるディスクでワークディレクトリを指定します。

Unicode モード

ソータトランスフォーメーションを最適化するには、PowerCenter Integration Service を ASCII モードで実行します。PowerCenter Integration Service が Unicode モードで実行される場合、ソータトランスフォーメーションは追加のデータをディスクに渡します。

Source Qualifier トランスフォーメーションの最適化

Integration Service がソースから一意な値を選択するようにしたい場合は、Source Qualifier トランスフォーメーションに対して Select Distinct オプションを使用します。Select Distinct オプションを使用して、データフローの初期段階で不必要なデータにフィルタをかけます。これによってパフォーマンスが向上します。

SQL トランスフォーメーションの最適化

SQL トランスフォーメーションの作成時には、トランスフォーメーション内で定義した外部 SQL クエリーを使用するように、トランスフォーメーションを設定します。SQL トランスフォーメーションをスクリプトモードで実行するように設定すると、Integration Service では、各入力行に対して外部 SQL スクリプトを処理します。トランスフォーメーションをクエリーモードで実行すると、Integration Service では、トランスフォーメーション内で定義した SQL クエリーを処理します。

Integration Service は、セッション内で新しいクエリーを処理するたびに、SQLPrepare という関数を呼び出し、SQL プロシージャを作成してデータベースに渡します。入力行ごとにクエリーが変更されると、パフォーマンスに影響します。

トランスフォーメーションをクエリーモードで実行すると、トランスフォーメーションに静的クエリーが作成され、パフォーマンスが改善されます。クエリー句内のデータが変更されても、静的クエリーステートメントは変更されません。静的クエリーを作成するには、SQL エディタで文字列の置換の代わりにパラメータのバインドを使用します。パラメータのバインドの使用時は、クエリー句内のパラメータを、トランスフォーメーション入力ポート内の値に設定します。

SQL クエリーにコミットおよびクエリーステートメントが含まれる場合、Integration Service では、コミットまたはロールバックするたびに SQL プロシージャを再作成する必要があります。パフォーマンスを最適化するためには、SQL トランスフォーメーションクエリーでトランザクションステートメントを使用しないでください。

SQL トランスフォーメーションの作成時には、トランスフォーメーションをデータベースに接続する方法を設定します。接続方法には、静的に接続する方法と、実行時に接続情報をトランスフォーメーションに渡す方法があります。

静的接続を使用するようにトランスフォーメーションを設定する場合は、Workflow Manager 接続から接続を 1 つ選択します。SQL トランスフォーメーションは、セッション中に一度だけデータベースに接続します。動的情報を渡すと、SQL トランスフォーメーションは、入力行を処理するたびにデータベースに接続します。

XML トランスフォーメーションの最適化

非投影グループは XML パーサートランスフォーメーションや XML ソース定義から除外してかまいません。これらの非投影グループにメモリを割り当てる必要はありませんが、プライマリキーと外部キーの制約は保持する必要があります。

トランスフォーメーションエラーの除去

トランスフォーメーションエラーが多数発生すると、Integration Service のパフォーマンスが低下します。それぞれのトランスフォーメーションエラーに対して、Integration Service は、一時停止してそのエラーを調べ、エラーの原因となっている行をデータフローから削除します。次に、通常は Integration Service のセッションログファイルにその行が書き込まれます。

Integration Service で変換エラー、競合しているマッピングロジック、およびエラーとして設定されている条件（Null 入力など）が検出されると、トランスフォーメーションエラーが発生します。トランスフォーメーションエラーが発生した箇所を調べるには、セッションログをチェックします。エラーが特定のトランスフォーメーションで発生している場合は、そのトランスフォーメーション制約を評価してください。

エラーしきい値を設定しない場合、Integration Service は、エラー行を継続して処理します。このため、セッションの実行時間が増加します。パフォーマンスを最適化するには、エラーしきい値を設定し、設定した数の行エラーが発生した場合にセッションを停止します。

大量のトランスフォーメーションエラーを発生させるセッションを実行する必要がある場合は、トレースレベルを低く設定することによってパフォーマンスを向上させることができます。ただし、この方法はトランスフォーメーションエラーに対する長期的な対策としてはお勧めできません。

関連項目：

- [「エラートレース」 \(ページ 48\)](#)

第 7 章

セッションの最適化

この章では、以下の項目について説明します。

- [グリッド, 43 ページ](#)
- [ブッシュダウンの最適化, 44 ページ](#)
- [コンカレントセッションおよびワークフロー, 44 ページ](#)
- [バッファメモリ, 44 ページ](#)
- [キャッシュ, 46 ページ](#)
- [ターゲットベースのコミット, 47 ページ](#)
- [リアルタイム処理, 48 ページ](#)
- [ステージングエリア, 48 ページ](#)
- [ログファイル, 48 ページ](#)
- [エラートレース, 48 ページ](#)
- [セッション実行後のメール, 49 ページ](#)

グリッド

グリッドを使用することで、セッションやワークフローのパフォーマンスを向上させることができます。グリッドは、一連のノードに対して割り当てられたエイリアスのことで、ノードに対するワークフローやセッションの割り当てを自動化することができます。

ロードバランサは、どのノードにも過負荷をかけることのないように、ノード間にタスクを分散します。

グリッドを使用すると、Integration Service はワークフロータスクおよびセッションスレッドを複数のノードに割り当てます。ロードバランサは、どのノードにも過負荷をかけることのないように、ノード間にタスクを分散します。グリッドのノード上で実行されているワークフローおよびセッションは、次のようなパフォーマンス向上を実現します。

- Integration Service の負荷を平均化する。
- コンカレントセッションのプロセスが速くなる。
- パーティションのプロセスが速くなる。

Integration Service は、入力データの解析および出力データのフォーマット設定に CPU リソースを必要とします。セッションの抽出/ロードステップにパフォーマンスボトルネックがある場合は、グリッドによるパフォーマンス向上が可能です。

メモリまたは一時記憶域がパフォーマンスボトルネックとなっている場合は、グリッドによるパフォーマンス向上が可能です。キャッシュメモリを持つトランスフォーメーションが PowerCenter マッピングに含まれて

いる場合は、十分なキャッシュメモリおよび各キャッシュインスタンス用の別個のディスクストレージを配置することにより、パフォーマンスを向上できます。

グリッド上でセッションを実行すると、グリッドにより提供されるセッション実行用リソースが増えるため、スループットを向上できます。セッション数が少数の場合はグリッド上で一括実行すれば、パフォーマンスが向上します。コンカレントセッションのパーティション数がノード数を下回る場合、グリッド上でのセッション実行のほうが、グリッド上でのワークフロー実行よりも効率的です。

グリッド上で複数のセッションを実行すると、セッションのサブタスクと他のコンカレントセッションのサブタスクとでノードリソースが共有されます。グリッド上でセッションを実行するには、別のいくつかのノード上で実行されているプロセス間の協調が必要です。マッピングによっては、グリッド上でセッションを実行するには、あるノードから別のノードへのデータ移動に追加のオーバーヘッドを必要とする場合があります。グリッド上での複数セッション実行時には、各ノード上にメモリおよび CPU リソースがロードされるだけでなく、ネットワークトラフィックが増大します。

グリッド上でのワークフロー実行時には、ノード間の協調を必要とすることなしに、ノード上にメモリおよび CPU リソースがロードされます。

関連項目：

- [「グリッドのデプロイメントの最適化」 \(ページ 50\)](#)

プッシュダウンの最適化

セッションのパフォーマンスを向上させるには、ソースデータベースまたはターゲットデータベースにトランスフォーメーションロジックを転送します。マッピングおよびセッションの設定に基づき、Integration Service はサービス内のトランスフォーメーションロジックを処理する代わりに、ソースやターゲットデータベースに対して SQL を実行します。

コンカレントセッションおよびワークフロー

可能な限り、パフォーマンスを向上させるためにセッションおよびワークフローを同時に実行します。たとえば、次元テーブルやファクトテーブルがある分析スキーマにデータをロードすると、次元テーブルが同時にロードされます。

バッファメモリ

統合サービスでは、セッションを初期化するときに、ソースデータとターゲットデータを保持するためのメモリブロックが割り当てられます。統合サービスは、ソースパーティションおよびターゲットパーティションごとに少なくとも2つのブロックを割り当てます。多数のソースやターゲットを使用するセッションでは、追加のメモリブロックが必要になる場合があります。統合サービスでソースデータ保持のための十分なメモリブロックを割り当てることができない場合、セッションは失敗します。

バッファメモリの量を設定したり、ランタイムのバッファ設定を計算するように統合サービスを設定したりすることができます。

空きメモリブロック数を増やすには、以下のセッションプロパティを調整します。

- **DTM バッファサイズ。**セッションプロパティの [プロパティ] タブにある DTM バッファサイズを増やします。
- **デフォルトのバッファブロックサイズ。**セッションプロパティの [設定オブジェクト] タブにあるバッファブロックサイズを減らします。

注: データのパーティション化が有効な場合、DTM バッファサイズは、すべてのパーティションに割り当てられたすべてのメモリバッファプールの合計サイズになります。 n 個のパーティションを含むセッションの場合、DTM バッファサイズは 1 つのパーティションだけのセッションの値に対して少なくとも n 倍に設定します。

DTM バッファサイズの拡大

[DTM バッファサイズ] の設定では、統合サービスが DTM バッファメモリとして使用するメモリの量を指定します。DTM バッファメモリを拡張すると、統合サービスで作成されるバッファブロックが増大するため、一時的な速度低下時のパフォーマンスが向上します。

DTM バッファメモリ割り当てを増やすと、一般的にパフォーマンスは最初に上昇した後で安定します。パフォーマンスが大きく向上しない場合は、DTM バッファメモリ割り当てがセッションのパフォーマンスに影響を及ぼしていないことが分かります。

DTM バッファサイズを増やすには、セッションプロパティを開いて [プロパティ] タブをクリックし、[パフォーマンス] 設定の DTM バッファサイズのプロパティを編集します。バッファブロックサイズの倍数で [DTM バッファサイズ] プロパティを増やします。

バッファブロックサイズの最適化

マシンが物理的なメモリを制限し、セッションのマッピングに多くのソース、ターゲット、パーティションがあるときには、バッファサイズを減らした方がよい場合もあります。

非常に大きなデータ行を操作する場合は、バッファブロックサイズを増やすとパフォーマンスが向上します。おおよその行サイズがわからない場合は、以下の手順を実行して行サイズを決定します。

必要なバッファサイズを評価するには：

1. Mapping Designer で、セッションのマッピングを開きます。
2. ターゲットインスタンスを開きます。
3. このトランスフォーメーションには以下のポートがあります。
4. ターゲット内の全カラムの精度を合計します。
5. マッピング内に複数のターゲットがある場合は、追加するターゲットごとに手順 2~4 を繰り返して、各ターゲットの精度を計算します。
6. マッピング内の各ソース定義に対して、手順 2~5 を繰り返します。
7. ソースおよびターゲットの精度すべての中から最大の精度を選択してバッファブロックサイズの合計の精度を求めてください。

合計の精度は、データの最大のレコード 1 つを移動するのに必要な合計のバイト数を表します。たとえば、合計の精度が 33,000 である場合、統合サービスがその行を移動するには 33,000 バイトのバッファブロックが必要になります。バッファブロックサイズが 64,000 バイトしかない場合、統合サービスは一度に 1 行しか移動できません。

バッファブロックサイズを設定するには、セッションプロパティを開いて [設定オブジェクト] タブをクリックし、[詳細設定] の [デフォルトのバッファブロックサイズ] プロパティを編集します。

DTM バッファメモリ割り当てと同様、バッファブロックサイズを増やしてもパフォーマンスは向上します。パフォーマンスが大きく向上しない場合は、バッファブロックサイズがセッションのパフォーマンスに影響を及ぼしていないことが分かります。

キャッシュ

Integration Service は、XML ターゲット、および Aggregator、Rank、Lookup、Joiner の各トランスフォーメーションに対してインデックスとデータキャッシュを使用します。Integration Service は、パイプラインに返す前に変換されたデータをデータキャッシュに格納します。グループ情報はインデックスキャッシュに格納されます。また、Integration Service はキャッシュを使用して、Sorter トランスフォーメーションのデータを格納します。

キャッシュメモリの量を設定するには、キャッシュカルキュレータを使用するか、またはキャッシュサイズを指定します。また、Integration Service を設定して、ランタイムにキャッシュメモリ設定を計算するようにすることもできます。

割り当て済みキャッシュがデータの格納に十分な大きさでない場合、Integration Service では一時ディスクファイル（キャッシュファイル）内にデータを格納することにより、セッションデータの処理を可能にします。パフォーマンスは、Integration Service が一時ファイルに対してページングを行うたびに低下します。パフォーマンスカウンタを検証し、Integration Service がどのくらいの頻度でファイルに対してページングしているかを確認します。

キャッシュを最適化するには、次のタスクを実行します。

- 接続される入出力ポートまたは出力ポートを制限する。
- キャッシュディレクトリに最適な場所を選択する。
- キャッシュサイズを増やす。
- 64 ビットバージョンの PowerCenter を使用して、大きいキャッシュセッションを実行する。

接続されるポート数の制限

データキャッシュを使用するトランスフォーメーションでは、接続する入出力ポートや出力ポートを制限します。入出力ポートや出力ポートの接続数を制限することで、トランスフォーメーションがデータキャッシュに格納するデータ量を減らします。

キャッシュディレクトリの場所

グリッドで Integration Service を実行し、一部の Integration Service ノードのみが共有キャッシュファイルディレクトリに高速アクセスしている場合は、ディレクトリへのアクセスが高速なノードで実行するように、大きいキャッシュを持つ各セッションを設定します。ディレクトリへのアクセスが高速なノードで実行するようにセッションを設定するには、以下の手順を実行します。

1. PowerCenter リソースを作成します。
2. ディレクトリへのアクセスが高速なノードで、そのリソースを使用できるようにします。
3. そのリソースをセッションに割り当てます。

グリッドのすべての Integration Service プロセスがキャッシュファイルに低速でアクセスしている場合は、Integration Service プロセスごとに別々のローカルキャッシュファイルディレクトリを設定します。キャッシュディレクトリを含むマシンと同一のマシンで Integration Service プロセスを実行する場合は、キャッシュファイルへのアクセスが高速になることもあります。

注: マップドライブまたはマウントドライブ上に大量のデータをキャッシュするときに、パフォーマンスの低下に関する問題に遭遇する場合があります。

キャッシュサイズの拡張

キャッシュサイズを設定して、トランスフォーメーションを処理するために割り当てられるメモリ量を指定します。設定するメモリ容量は、使用するメモリキャッシュおよびディスクキャッシュのサイズによって異なります。設定したキャッシュサイズが、メモリでトランスフォーメーションを処理するのに十分な大きさではない場合は、Integration Service ではトランスフォーメーションの一部はメモリで実行され、残りのトランスフォーメーションを処理するためにキャッシュファイルに情報がページングされます。パフォーマンスは、Integration Service が一時ファイルに対してページングを行うたびに低下します。

セッションのパフォーマンス詳細を調べて、Integration Service がキャッシュファイルにいつページングしているかを特定します。Aggregator、Rank、または Joiner トランスフォーメーションの *Transformation_readfromdisk* カウンタまたは *Transformation_writetodisk* カウンタには、トランスフォーメーションを処理するために Integration Service がディスクに対して何回ページングするかが示されます。

セッション内にキャッシュを使用するトランスフォーメーションがあり、十分にメモリのあるマシンでセッションを実行する場合は、キャッシュサイズを増やして、トランスフォーメーションがメモリで処理されるようにします。

64 ビットバージョンの PowerCenter の使用

大量のデータを処理したり、メモリ集約型トランスフォーメーションを実行する場合、64 ビットの PowerCenter バージョンを使用してセッションのパフォーマンスを向上させることができます。64 ビットバージョンには大型のメモリスペースがあり、ディスクの入出力を劇的に削減または除去することができます。

これにより、以下の領域でのセッションパフォーマンスを改善できます。

- **キャッシュ。** 64 ビットプラットフォームにより、Integration Service は、32 ビットプラットフォームの 2 GB キャッシュ制限による制約を受けません。
- **データスループット。** 利用可能な大型のメモリスペースにより、reader や writer、DTM スレッドは大型のデータブロックを処理することができます。

ターゲットベースのコミット

コミット間隔の設定は、Integration Service がターゲットにデータをコミットするポイントを指定するためのものです。Integration Service がコミットするたびに、パフォーマンスは低下します。したがって、コミット間隔が短いほど Integration Service はターゲットデータベースに頻繁に書き込みを行い、その結果、全体のパフォーマンスが低下します。

コミット間隔を長く設定すると、Integration Service がコミットする回数が減り、パフォーマンスが向上します。

コミット間隔を長くする場合は、ターゲットデータベースでのログファイルの制限を考慮に入れてください。コミット間隔が長すぎる場合、Integration Service がデータベースログファイルの最大サイズまで書き込みを行って、その結果、セッションの失敗を引き起こす可能性があります。

そのため、コミット間隔を長くした場合の利点と、失敗したセッションのリカバリに要する時間とのどちらを重視するかを比較検討する必要があります。

セッションプロパティの [全般オプション] 設定をクリックして、コミット間隔を確認し、調整します。

リアルタイム処理

フラッシュ待ち時間

フラッシュ待ち時間は、Integration Service でソースからリアルタイムデータをフラッシュする頻度を決定します。フラッシュ待ち時間の間隔を短く設定すると、Integration Service がターゲットにメッセージをコミットする頻度が増えます。Integration Service がターゲットにメッセージをコミットするたびに、セッションでリソースが消費されてスループットが低下します。

スループットを改善するには、フラッシュ待ち時間を長くします。ハードウェアおよび使用可能なリソースに従って、特定のしきい値までフラッシュ待ち時間を大きくすると、スループットが向上します。

ソースベースのコミット

ソースベースのコミット間隔は、Integration Service でターゲットにリアルタイムデータをコミットする頻度を決定します。待ち時間を最短にするには、ソースベースのコミットを 1 に設定します。

ステージングエリア

ステージング領域を使用すると、Integration Service はデータに対して複数のパスを実行します。可能であれば、ステージングエリアを削除してパフォーマンスを向上させます。Integration Service は 1 回のパスで複数のソースを読み出すことができるため、ステージングエリアの必要性はそれほど大きくありません。

関連項目：

- [「Single-Pass 読み込みの設定」 \(ページ 28\)](#)

ログファイル

セッションログファイルおよびワークフローログファイルに書き込まないようにワークフローを設定すると、ワークフローの実行が高速化されます。ワークフローおよびセッションでは、常にバイナリログが作成されます。ログファイルに書き込むようにセッションまたはワークフローを設定すると、Integration Service によってロギングイベントが 2 回書き込まれます。バイナリログセッションおよびワークフローログには、管理者ツールからアクセスできます。

エラートレース

パフォーマンスを改善するには、Integration Service がセッションを実行するときに生成されるログイベントの数を削減します。セッション内に大量のトランスフォーメーションエラーがあっても、修正する必要がない場合は、トレースレベルを [Terse] に設定します。このトレースレベルの場合、Integration Service はエラーメッセージやリジェクトデータの行レベル情報を書き込みません。

マッピングをデバッグする必要があり、トレースレベルを [Verbose] に設定すると、セッションを実行した際にパフォーマンスの著しい低下を体験する可能性があります。パフォーマンスのチューニングを行っているときは、[Verbose] を使用しないでください。

セッションのトレースレベルは、マッピング内のトランスフォーメーション固有のトレースレベルをオーバーライドします。ただし、この方法は多くのトランスフォーメーションエラーが発生する場合の長期的な対策としてはお勧めできません。

セッション実行後のメール

セッション後に発信される電子メールにセッションログを添付すると、フラットファイルのロギングが有効になります。フラットファイルのロギングを有効にすると、Integration Service はディスクからセッションログファイルを取得します。フラットファイルのロギングを有効にしない場合、Integration Service は Log Manager からログイベントを取得し、電子メールに添付するセッションログファイルを生成します。Integration Service がログサービスからセッションログを取得する場合は、特にセッションログファイルが大きく、マスター DTM とは異なるノードでログサービスが実行されているときに、ワークフローのパフォーマンスが低下します。最高のパフォーマンスを実現するには、セッションログを添付するためにセッション後のメールを設定するときに、セッションによるログファイルへの書き込みが行われるように設定します。

第 8 章

グリッドのデプロイメントの最適化

この章では、以下の項目について説明します。

- [グリッドのデプロイメントの最適化の概要, 50 ページ](#)
- [ファイルの格納, 50 ページ](#)
- [共有ファイルシステムの使用, 52 ページ](#)
- [ファイルシステム全体へのファイルの分散, 54 ページ](#)
- [Sequence Generator トランスフォーメーションの最適化, 55 ページ](#)

グリッドのデプロイメントの最適化の概要

グリッド上で PowerCenter を実行するとき、リソースを効率的に使用し、拡張性を最大限まで活用するように、グリッド、セッション、ワークフローを設定できます。

グリッドの PowerCenter のパフォーマンスを改善するには、以下のタスクを実行します。

- グリッドへのノードの追加
- ストレージ容量および帯域幅の拡張
- 共有ファイルシステムの使用
- 次のタスク完了時に高スループットのネットワークを使用
 - ネットワーク経由でのソースおよびターゲットへのアクセス
 - [グリッド上のセッション] オプション使用時に、グリッドのノード間でデータを転送

ファイルの格納

PowerCenter をグリッドで実行するように設定する場合は、ソースファイル、ログファイルおよびキャッシュファイルなど、さまざまなタイプのセッションファイルの格納場所を指定します。パフォーマンスを改善するには、最適な場所にファイルを格納します。たとえば、永続キャッシュファイルは、高帯域幅の共有ファイルシステムに格納します。ファイルタイプが異なれば、格納要件も異なります。

ファイルを格納できる場所のタイプは、以下のとおりです。

- **共有ファイルシステム。**すべての Integration Service プロセスが同じファイルにアクセスできるように、ファイルを共有ファイルシステムに格納します。ファイルを低帯域幅および高帯域幅共有ファイルシステムに格納できます。
- **ローカル。**ほかの Integration Service プロセスからファイルへのアクセスが必要ないときは、そのファイルは Integration Service プロセスを実行しているローカルマシンに格納します。

高帯域幅共有ファイルシステムのファイル

以下のファイルはセッション中に頻繁にアクセスされることがあるため、高帯域幅共有ファイルシステムに格納します。

- ソースファイル（ルックアップ用のフラットファイルを含む）。
- ターゲットファイル（パーティション化されたセッション用の統合ファイルを含む）。
- ルックアップまたは差分集計用の永続キャッシュファイル。
- グリッド上のグリッド対応セッション専用¹の非永続キャッシュファイル。

これにより、Integration Service はキャッシュを 1 回だけ構築できます。これらのキャッシュファイルをローカルファイルシステムに格納すれば、Integration Service はパーティショングループごとにキャッシュを構築します。

低帯域幅共有ファイルシステムのファイル

以下のファイルはセッション中にアクセスされる回数が少ないため、低帯域幅共有ファイルシステムに格納します。

- パラメータファイルまたは他の設定関連のファイル。
- 間接ソースファイルまたは間接ターゲットファイル。
- ログファイル。

ローカルストレージファイル

共有ファイルシステム使用時に不要なファイル共有を避けるために、以下のファイルはローカルで格納します。

- グリッドに非対応なセッション用の非永続キャッシュファイル（Sorter トランスフォーメーションの一時ファイルを含む）。
- パーティション化されたセッション用に順次統合を実行する場合の、さまざまなパーティション用の個別ターゲットファイル。
- セッション実行の最後に削除されるその他の一時ファイル。通常、これを実現するには、ローカルファイルシステムに \$PmTempFileDir を設定します。

帯域幅が高いときでも、これらのファイルを共有ファイルシステムに格納することは避けます。

共有ファイルシステムの使用

ファイル共有には、以下の共有ファイルシステムを使用できます。

- Windows の CIFS (SMB) または UNIX の Network File System (NFS) などのネットワークファイルシステム。ネットワークファイルシステムは高パフォーマンスの計算向けではありませんが、シーケンシャルファイルアクセスにもうまく機能します。
- クラスタファイルシステム。クラスタファイルシステムでは、高帯域幅のファイルアクセスを可能にするノードグループの他に、ファイルおよびディレクトリ用の統合された名前空間を提供します。クラスタファイルシステムのパフォーマンスは、直接接続されたローカルファイルシステムに似ています。

注: 高可用性オプションがある場合には、クラスタファイルシステムを使用します。

適正な設定とチューニングがグリッドのわずかなパフォーマンスにとって重要となります。また、マッピングとセッションの設定によって、共有ファイルシステムの本質的な限界を回避できます。

共有ファイルシステムの設定

以下の一般ガイドラインに従って、共有ファイルシステムを設定します。

- ネットワークは十分な帯域幅を確保する。
- 内在するストレージが十分な I/O 帯域幅を確保する。
- 共有ファイルシステムのデーモンを、特にクライアントを、ファイルアクセスを迅速にするだけの十分なスレッドを持つように設定する。たとえば、IBM は、同時アクセスを必要とするファイル数を見積もり、各ファイルに最低 2 個の biod スレッドを与えることを推奨しています。

フラットファイルソースまたはターゲットを使用するグリッド上で同時セッションを実行するときは、同時に必要とするソースまたはターゲットファイルに各パーティションがアクセスできるように、十分なスレッドを用意します。

- アクセス要件に基づいて、共有ファイルシステムのマウントポイントを設定する。フラットファイルソースまたはターゲットを使用するグリッド上でシーケンシャルセッションを実行するときは、デフォルトの先読みプロセスまたは後書きプロセスの有効性を低下させる可能性のある設定を回避します。ファイルシステムは、先読みおよび後書きでシーケンシャルファイルアクセスを最適化します。
- 必要に応じて、共有ファイルシステムの先読みおよび後書きの設定をチューニングする。
- クライアントとサーバーの両方の共有ファイルシステムのキャッシュ設定を検討する。デフォルトの設定を増大すると、パフォーマンスを増大することができます。
- データアクセス後にメモリページを解放するには、ファイルシステムの「release-behind」値を設定します。そうしなければ、大きいファイルの読み出しまたは書き込み時に、システムパフォーマンス低下の可能性がありえます。
- アクセスパターンの違いにより、ソースおよびターゲット、永続キャッシュに対して異なるマウントポイント使用できます。

詳細については、共有ファイルシステムのマニュアルを参照してください。

CPU とメモリの使用量の均衡化

ローカルファイルシステムとは異なり、共有ファイルシステムサーバーでは、追加の CPU サイクルを使用してファイルにアクセスできます。計算ノードの 1 つを残りのノードの共有ファイルシステムサーバーとして使用すると、そのノードが過負荷となってグリッド全体のボトルネックとなるおそれがあります。共有ファイルシステムサーバーが過負荷状態になると、反復転送およびタイムアウト要求と同様に、CPU サイクルが増大します。

これを回避するために、PowerCenter グリッドノード専用の共有ファイルシステムサーバーとして、1 台以上のマシンを使用します。各マシンは、要求されるタスクに対応した十分なストレージ、CPU 数およびネットワーク帯域幅を持つ必要があります。

また、共有ファイルシステムサーバーをクロスマウントして、ファイルサーバーの負荷をグリッドのノード全体に分散できます。I/O と CPU の使用率が均衡するように PowerCenter のマッピングとセッションを設定する場合は、共有ファイルシステムサーバーのクロスマウントによってパフォーマンスを最適化できます。グリッド内のノード数が小さく、I/O と CPU の使用率がつり合っている場合は、専用の共有ファイルシステムサーバーは必要とならないことがあります。

専用のまたはクロスマウントされた複数の共有ファイルシステムサーバーを使用するとき、サーバー全体に共有ファイルを分散するようにします。

関連項目：

- [「ファイルシステム全体へのファイルの分散」 \(ページ 54\)](#)

PowerCenter マッピングとセッションの設定

パフォーマンスを向上させる最も重要な方法の 1 つは、不要なファイル共有を避けることです。設定が適切であると、共有ファイルシステムにおけるソースおよびターゲットファイルのシーケンシャルアクセスは十分なパフォーマンスを実現します。ただし、永続キャッシュファイル、特に大きい永続キャッシュファイルに要求されるランダムアクセスは問題となることがあります。

共有ファイルシステムを備えたグリッドの永続的なダイナミックルックアップなどの永続キャッシュファイルを設定するには、以下のガイドラインに従います。

- できれば、より小さい永続キャッシュファイルをメモリ内で維持するように、セッションのキャッシュサイズを設定する。
- Sorter トランスフォーメーションをマッピングに追加して、永続的なルックアップの前に入力行をソートする。作業を永続ルックアップから Sorter トランスフォーメーションに移行することにより、Sorter トランスフォーメーションはローカルファイルシステムを使用できるので、パフォーマンスを向上できます。
- ルックアップキャッシュの同じページへのアクセスを必要とする行をグループ化して、Integration Service がキャッシュの各ページを読む回数を最小に抑える。
- 入力データのサイズが大きいときは、入力データの管理にソーススペースのコミットを使用して、メモリ内でソートが実行できるようにする。

たとえば、マッピングロジックの変更がなければ縮小できない永続ダイナミックルックアップが 4 GB、またソースデータが 10 GB とします。最初に、Sorter トランスフォーメーションを追加して入力データをソートし、ルックアップキャッシュのランダムアクセスを削減し、その後、以下のタスクを完了します。

- ソーススペースのコミットを 1 GB のコミット間隔で実行するように、セッションを設定する。
- Sorter トランスフォーメーションのトランザクション範囲を「トランザクション」に設定する。
- ソース入力に十分な 1 GB キャッシュサイズに合わせて、Sorter トランスフォーメーションを設定する。

この設定では、Integration Service は、一度に 1 GB の入力データをソートし、キャッシュ内の似たデータへのアクセスを必要とする行を永続ルックアップへ渡します。

- 複数のファイルシステムが使用可能な場合は、個々のファイルシステムを使用するために、各パーティションのキャッシュファイルを設定します。
- 複数のファイルシステムが使用可能な場合は、ファイルを別々のファイルシステムに分散するようにセッションを設定します。

ファイルシステム全体へのファイルの分散

各ファイルシステムが独立した物理ディスクサブシステムを使用することを想定し、ファイルシステムの結合した帯域幅を使用するために、ファイルを異なるファイルシステムに分散させます。別のファイルシステムにファイルを分散することにより、共有ファイルシステムまたは対称型マルチプロセッシング（SMP）のいずれかを使用するグリッド上で、パフォーマンスを向上させることができます。

最適な I/O 帯域幅のために、複数のストレージデバイスにわたってファイルを分散するファイルシステムを選択します。クラスタファイルシステムを使用する場合は、ファイルをサーバー間で分散させます。可能な場合には、ソース、ターゲット、キャッシュの各ファイルを別々のストレージデバイスに配置します。

以下のガイドラインに従って、ファイルをファイルシステムに分散させます。

- **ソースファイル。** Integration Service が大量のファイルからデータを読み出せるよう、ファイルシステムにソースファイルを配置する場合は、ファイルシステムの先読み設定を、大きなファイルのキャッシュ前にチューニングします。
- **一時ファイル。** Integration Service が大きなファイルからデータを読み出して一時ファイルに書き込めるよう、ファイルシステムに一時ファイルを配置する場合は、大きなファイルに対するファイルシステムの読み書き設定をチューニングします。
- **ターゲットファイル。** Integration Service が大きなファイルをディスクに書き込めるよう、ファイルシステムにターゲットファイルを配置する場合は、大きなブロックの同時書き込み用にファイルシステムをチューニングします。ターゲットファイルには、パーティション化されたセッション用の統合ファイルを含めることができます。グリッド上のパーティション化されたセッションはファイルをディスクに書き込む必要があるため、ロックパフォーマンスが最適になるよう、ファイルシステムをチューニングします。

ファイルを分散するようにセッションを設定

ファイルの負荷を分散するようにセッションを手動で設定できます。負荷が大幅に変化するとき、あるいはクロスマウントされた共有ファイルシステムを持つグリッドへの新しいノードの追加を含めて、新しいセッションまたはファイルシステムを追加するとき、セッションを編集する必要があります。

セッションを手動で編集する代わりに、セッション変数を使用して、ファイルを別のディレクトリへ分散します。これにより、セッションファイルを必要に応じて別のファイルサーバーにリダイレクトできます。

セッション変数を使用するには以下のガイドラインを使用します。

- ビジネスロジックを表わすように、セッションのファイル名およびディレクトリ用の変数に名前を付ける。
- パラメータファイル内で、使用可能なすべてのファイルシステム全体にファイル負荷が均一に分散するように、各変数を定義する。また、ノード特有の変数も定義できます。
- 必要に応じて、パラメータファイルを処理するスクリプトを使用して、再設定を自動化する。

注: スクリプトを使用するとき、スクリプトで必要に応じてセッション変数を再定義できるように、パラメータファイルでプレースホルダを使用します。

パラメータファイルとスクリプトのガイドライン

パラメータファイルおよびスクリプトを作成する場合は、以下のガイドラインに従います。

- セッションファイルの場所の柔軟性と制御を容易に維持するには、スクリプトを使用してパラメータファイル内のプレースホルダを置き換える。
- ファイルの場所を定義するとき、推定のファイルサイズとファイルシステム容量を検討する。
- セッションおよびワークフローがビジネスに関係するファイルに同時にアクセスする必要がある場合は、ビジネスロジックに従ったファイルの整理を避ける。たとえば、カリフォルニアのファイルのあるファイル

システムに格納し、ニューヨークのファイルを別のファイルシステムに格納した場合は、セッションが両方のファイルに同時にアクセスする必要があるときにボトルネックが発生することがあります。

- できる限り、同じソース、ターゲットまたはルックアップの異なるパーティション用のファイルを別の複数のファイルシステムに格納する。

例

次の生のパラメータファイルの抜粋では、プレースホルダ“{fs}”は、ディレクトリが配置されるファイルシステムを表わし、使用される前にスクリプトによってそのファイルシステムを割り当てする必要があります。

```
[SessionFFSrc_FFTgt_CA]
  $InputFile_driverInfo_CA={fs}/driverinfo_ca.dat
  $SubDir_processed_CA={fs}
# Session has Output file directory set to:
# $PmTargetFileDir/$SubDir_processed_CA
# This file is the input of SessionFFSrc_DBTgt_CA.
  $SubDir_RecordLkup_Cache_CA={fs}
# This session builds this persistent lookup cache to be used
  # by SessionFFSrc_DBTgt_CA.
# The Lookup cache directory name in the session is set to:
# $PmCacheDir/$SubDir_RecordLkup_Cache_CA
[SessionFFSrc_FFTgt_NY]
  $InputFile_driverInfo_NY={fs}/driverinfo_ny.dat
  $SubDir_processed_NY={fs}
[SessionFFSrc_DBTgt_CA]
  $SubDir_processed_CA={fs}
# session has Source file directory set to:
# $PmTargetFileDir/$SubDir_processed_CA
  $SubDir_RecordLkup_Cache_CA={fs}
# Use the persistent lookup cache built in SessionFFSrc_FFTgt_CA.
```

次のパラメータファイルの抜粋では、プレースホルダが file_system_1 と file_system_2 などの該当するファイルシステム名にスクリプトによって置き換わります。

```
[SessionFFSrc_FFTgt_CA]
  $InputFile_driverInfo_CA=file_system_1/driverinfo_ca.dat
  $SubDir_processed_CA=file_system_2
# Session has Output file directory set to:
# $PmTargetFileDir/$SubDir_processed_CA
# This file is the input of SessionFFSrc_DBTgt_CA.
  $SubDir_RecordLkup_Cache_CA=file_system_1
# This session builds this persistent lookup cache to be used
  # by SessionFFSrc_DBTgt_CA.
# The Lookup cache directory name in the session is set to:
# $PmCacheDir/$SubDir_RecordLkup_Cache_CA
[SessionFFSrc_FFTgt_NY]
  $InputFile_driverInfo_NY=file_system_2/driverinfo_ny.dat
  $SubDir_processed_NY=file_system_1
[SessionFFSrc_DBTgt_CA]
  $SubDir_processed_CA=file_system_1
# session has Source file directory set to:
# $PmTargetFileDir/$SubDir_processed_CA
  $SubDir_RecordLkup_Cache_CA=file_system_2
# Use the persistent lookup cache built in SessionFFSrc_FFTgt_CA.
```

Sequence Generator トランスフォーメーションの最適化

Sequence Generator トランスフォーメーションを使用してグリッドでセッションを実行するときのパフォーマンスを向上させるには、キャッシュ値の数をデータの各行と同じ数まで増やします。この結果、マスター

DTM プロセスおよびワーカ DTM プロセスとリポジトリの間の通信が削減されます。マスター DTM およびワーカ DTM では、1 キャッシュ値につき 1 回の通信を行います。

たとえば、150,000 行のデータおよび 7 つの Sequence Generator トランスフォーメーションがあると想定します。キャッシュ値の数は 10 です。マスター DTM およびワーカ DTM では、15,000 回の通信を行います。キャッシュ値の数を 15,000 に増やすと、マスター DTM およびワーカ DTM の通信は 10 回になります。

第 9 章

PowerCenter コンポーネントの最適化

この章では、以下の項目について説明します。

- [PowerCenter コンポーネントの最適化の概要, 57 ページ](#)
- [PowerCenter リポジトリのパフォーマンスの最適化, 57 ページ](#)
- [Integration Service のパフォーマンスの最適化, 60 ページ](#)

PowerCenter コンポーネントの最適化の概要

次の PowerCenter コンポーネントのパフォーマンスを最適化することができます。

- PowerCenter リポジトリ
- Integration Service

PowerCenter を複数のマシンで実行する場合、リポジトリサービスと Integration Service は個別のマシンで実行します。大量のデータをロードする場合、処理能力の高いマシンで Integration Service を実行します。また、リポジトリサービスは PowerCenter リポジトリをホスティングするマシンで実行します。

PowerCenter リポジトリのパフォーマンスの最適化

PowerCenter リポジトリのパフォーマンスを最適化する次のタスクを実行します。

- PowerCenter リポジトリがリポジトリサービスプロセスと同じマシンにあることを確認します。
- オブジェクトクエリーの条件を整理します。
- DB2 データベースにインストールした場合は、PowerCenter リポジトリの単一ノードのテーブルスペースを使用します。
- リポジトリを DB2 データベースまたは Microsoft SQL Server データベースにインストールした場合は、PowerCenter リポジトリのデータベーススキーマを最適化します。

リポジットリサービスプロセスおよびリポジットリの場合

高可用性オプションを設定していないリポジットリサービスのパフォーマンスを最適化することができます。パフォーマンスを最適化するには、リポジットリデータベースを格納したマシンでリポジットリサービスプロセスを実行中であることを確認します。

オブジェクトクエリの条件の整理

リポジットリサービスは複数の条件が設定されているパラメータを入力順に処理します。予想通りの結果を得るため、また、パフォーマンスを向上させるためには、実行する順番にパラメータを入力してください。

単一ノードの DB2 データベースのテーブルスペース

PowerCenter リポジットリを単一ノードのテーブル領域に格納する場合、IBM DB2 EEE データベースのリポジットリ性能を最適化することができます。IBM DB2 EEE データベースを設定する際に、データベース管理者は単一ノードでデータベースを定義することができます。

PowerCenter クライアントおよび Integration Service は、テーブル領域に 1 つのノードが含まれる場合、リポジットリテーブルが別々のデータベースノードに置かれている場合よりも素早くリポジットリにアクセスできます。

リポジットリの作成、コピー、リストアを行う際にテーブル領域名を指定しない場合は、DB2 システムにより各リポジットリテーブルにデフォルトのテーブル領域が指定されます。DB2 システムで単一ノードのテーブル領域が指定される場合もあれば、指定されない場合もあります。

データベーススキーマの最適化

Administration Console でリポジットリサービスのデータベーススキーマの最適化オプションを有効にすると、IBM DB2 データベースおよび Microsoft SQL Server データベース上のリポジットリのパフォーマンスを改善できます。データベーススキーマの最適化オプションを有効にすると、リポジットリサービスでは、可能な場合は常に可変長文字データを CLOB 列ではなく Varchar(2000)列に格納します。Varchar(2000)列を使用すると、以下のようにリポジットリのパフォーマンスが改善されます。

- **ディスクアクセスの減少。**PowerCenter リポジットリでは、データベーステーブル内の列に Varchar データを直接格納します。CLOB データの場合は、別のテーブルに参照として格納されます。リポジットリから CLOB データを取得するために、リポジットリサービスでは、1 つのデータベーステーブルにアクセスして参照を取得してから、参照先のテーブルにアクセスしてデータを読み込む必要があります。Varchar データを取得するために、リポジットリサービスでは 1 つのデータベーステーブルにアクセスします。
- **キャッシングの改善。**リポジットリデータベースバッファマネージャでは、Varchar 列はキャッシュできませんが、CLOB 列はキャッシュできません。

データベーススキーマを最適化すると、以下の操作のリポジットリパフォーマンスを改善できます。

- リポジットリのバックアップ
- リポジットリのリストア
- リポジットリオブジェクトのエクスポート
- オブジェクト間の依存関係の一覧表示
- フォルダの配置

通常、リポジットリデータベースおよびページサイズが大きくなると、パフォーマンスもそれに応じて改善されます。したがって、大規模な PowerCenter リポジットリでは、データベーススキーマを最適化すると、より大幅にパフォーマンスを改善できます。

リポジトリコンテンツを作成するか、既存のリポジトリをバックアップおよびリストアする場合は、データベーススキーマを最適化できます。データベーススキーマを最適化するには、リポジトリデータベースが以下のページサイズ要件を満たす必要があります。

- **IBM DB2。** データベースのページサイズが 4KB 以上。 ページサイズが 16KB 以上ある一時テーブルスペースを少なくとも 1 つ。
- **Microsoft SQL Server。** データベースのページサイズが 8KB 以上。

リポジトリサービスにおけるオブジェクトキャッシュ

リポジトリオブジェクトのキャッシュにより、ワークフローを実行するときのパフォーマンスが向上します。リポジトリサービスがオブジェクトをメモリにキャッシュすると、統合サービスは、ワークフローの実行を完了する必要があるキャッシュされたオブジェクトに、より簡単にアクセスできます。

リポジトリサービスのオブジェクトのキャッシュを管理するには、次のプロパティを設定します。

プロパティ	説明
リポジトリエージェントキャッシング	オプション。リポジトリエージェントキャッシングを有効にします。これにより、統合サービスが複数のセッションを繰り返し実行する場合のパフォーマンスが向上します。このプロパティを有効にすると、リポジトリサービスのプロセスにより、統合サービスが要求するメタデータとリポジトリオブジェクトを記述するメタデータがキャッシュされます。デフォルトは [Yes] です。
エージェントキャッシュ容量	オプション。リポジトリエージェントキャッシングを有効にしたときに、キャッシュが含むことができるオブジェクトの数。リポジトリサービスプロセスを実行しているマシンに利用可能なメモリがある場合は、オブジェクト数を増やすことができます。デフォルトは 10,000 です。
エージェントキャッシュで書き込みを許可	オプション。リポジトリエージェントキャッシングを有効にすると、PowerCenter Client ツールを使用してリポジトリ内のメタデータを変更することができます。書き込みを有効にした場合は、リポジトリサービスにより、PowerCenter Client でメタデータを保存するたびにキャッシュがフラッシュされます。書き込みを無効にすると、PowerCenter Client を介してリポジトリにメタデータを保存することはできず、キャッシュはフラッシュされません。書き込みを無効にした場合でも、統合サービスはリポジトリにセッションとワークフローのメタデータを書き込むことができます。統合サービスがメタデータを書き込む場合、リポジトリサービスはキャッシュをフラッシュしません。デフォルトは [Yes] です。

レジリエンスの最適化

リポジトリのパフォーマンスを向上させるには、アプリケーションサービスの、以下のレジリエンス関連のプロパティを設定します。

プロパティ	説明
レジリエンスタイムアウトの制限	オプション。サービスが、レジリエンスのためにリソースを確保する最大時間です。このプロパティにより、サービスに接続するクライアント上で制限が設定されます。制限を超えるすべてのレジリエンスタイムアウトは、制限で切り捨てられます。このプロパティの値が空白の場合、値はドメインレベル設定から取得されます。デフォルトは空白です。
レジリエンスタイムアウト	オプション。別のサービスへの接続の確立または再確立をサービスが試行する時間間隔。このプロパティの値が空白の場合、値はドメインレベル設定から取得されます。デフォルトは空白です。

Integration Service のパフォーマンスの最適化

Integration Service のパフォーマンスを最適化する次のタスクを実行します。

- Integration Service では、ODBC ドライバの代わりにネイティブドライバを使用します。
- 文字データが 7 ビット ASCII または EBCDIC の場合、Integration Service を ASCII データ移動モードで実行します。
- リポジトリサービスの PowerCenter メタデータをキャッシュします。
- Integration Service を高可用性で実行します。

注: Integration Service を高可用性に設定した場合、Integration Service はワークフローおよびセッションをリカバリしますが、一時的なネットワーク障害やマシンの故障により失敗することもあります。ワークフローまたはセッションからリカバリする場合、Integration Service は各ワークフローおよびセッションの状態を共有ディレクトリにある一時ファイルに書き込みます。これにより、パフォーマンスが低下する可能性があります。

ネイティブドライバおよび ODBC ドライバの使用

Integration Service では、ODBC ドライバまたはネイティブドライバを使用してデータベースに接続することができます。ネイティブドライバを使用して、パフォーマンスを向上させます。

ASCII データ移動モードでの Integration Service の実行

Integration Service によって処理される文字データがすべて 7 ビット ASCII または EBCDIC である場合は、Integration Service を ASCII データ移動モードで実行するように設定します。ASCII モードの場合は、Integration Service では 1 文字を 1 バイトとして保存します。Unicode モードで Integration Service を実行した場合は、1 文字に 2 バイトが使用されるため、セッションのパフォーマンスが低下することがあります。

リポジトリサービスの PowerCenter メタデータのキャッシュ

Repository Agent キャッシュを使用して、DTM プロセスのパフォーマンスを向上することができます。Repository Agent キャッシュを有効にすると、リポジトリサービスは Integration Service がリクエストしたメタデータをキャッシュします。メタデータをキャッシュすると、Integration Service はリポジトリからメタデータを取得する代わりに、タスク実行以降のキャッシュを読み込みます。キャッシュされるのは、Integration Service が要求したメタデータのみです。

たとえば、1,000 セッションのワークフローを実行するとします。このとき、キャッシュが有効になってから初めてワークフローを実行すると、Integration Service はリポジトリからメタデータを取得しますが、それ以降のワークフローでは、リポジトリサービスはキャッシュされたセッションのメタデータを取得するようになります。これにより、DTM プロセスのパフォーマンスが向上します。

第 10 章

システムの最適化

この章では、以下の項目について説明します。

- [システムの最適化の概要, 61 ページ](#)
- [ネットワークの速度の向上, 62 ページ](#)
- [複数の CPU の使用, 62 ページ](#)
- [ページングの削減, 62 ページ](#)
- [プロセッサバインドの使用, 63 ページ](#)

システムの最適化の概要

セッションが非効率的な接続や過負荷の Integration Service プロセスシステムに依存していると、パフォーマンスの低下が頻繁に発生します。システムの遅延は、ルータ、スイッチ、ネットワークプロトコル、および同時利用者数の過多が原因で生じることもあります。

ソースおよびターゲットのデータベース、ソースおよびターゲットのファイルシステム、およびドメイン内のノードが低速であると、セッションのパフォーマンスが低下する場合があります。システム管理者に依頼して、使用しているマシンのハードディスクを評価してもらってください。

システム監視ツールを使用してシステムにボトルネックがあると判断した場合、以下のグローバルな変更を加えることによって、すべてのセッションのパフォーマンスを向上させることができます。

- **ネットワーク速度の改善。** ネットワーク接続速度が遅いと、セッションのパフォーマンスが低下することがあります。ネットワークの応答性が良好かどうかをシステム管理者に調べてもらってください。Integration Service プロセスとデータベースの間のネットワークホップ数を減らします。
- **複数の CPU を使用する。** 複数の CPU を使用すると、複数のセッションや複数のパイプラインパーティションを並列で実行できます。
- **ページングの削減。** オペレーティングシステムの物理メモリが不足した場合は、物理メモリを解放するためにディスクへのページングが開始されます。ディスクへのページングが最小限になるように、Integration Service プロセスマシン用の物理メモリを構成します。
- **プロセッサバインドの使用。** マルチプロセッサ UNIX 環境では、Integration Service が大量のシステムリソースを使用する場合があります。プロセッサバインドを使用して、Integration Service プロセスによるプロセッサの使用を制御します。また、ソースデータベースとターゲットデータベースが同じマシン上にある場合は、プロセッサバインドを使用して、データベースが使用するリソースを制限します。

ネットワークの速度の向上

Integration Service のパフォーマンスは、ネットワーク接続に影響されます。ローカルディスクでは、ネットワークよりも 5~20 倍の速度でデータを転送できます。ネットワークアクティビティを最小限に抑えて Integration Service のパフォーマンス向上を図るため、以下の事項を検討します。

セッションでフラットファイルをソースまたはターゲットとして使用し、Integration Service が単一ノード上で実行されている場合は、Integration Service と同じマシン上にファイルを保存して、パフォーマンスを向上させます。フラットファイルを Integration Service 以外のマシン上に格納すると、セッションのパフォーマンスはネットワーク接続のパフォーマンスに依存するようになります。ファイルを Integration Service プロセスシステム上に移動し、ディスク領域を追加することで、パフォーマンスを向上させることができます。

リレーショナルソースまたはリレーショナルターゲットのデータベースを使用する場合は、ソースおよびターゲットデータベースと Integration Service プロセスの間のネットワークホップ数を最小限にするようにします。ターゲットデータベースをサーバーシステム上に移動することで、Integration Service のパフォーマンスを向上させることができます。

複数のパーティションを含むセッションを実行しているときは、ネットワーク管理者に依頼して、ネットワークを分析し、ネットワーク全体ですべてのパーティションからのデータ移動を扱うのに十分なバンド幅が確保されていることを確認してもらってください。

複数の CPU の使用

追加の CPU を使用できるようにシステムを設定して、パフォーマンスを向上させます。複数の CPU を使用すると、システムは複数のセッションや複数のパイプラインパーティションを並列で実行できます。

ただし、CPU を追加すると、ディスクにボトルネックが発生する可能性があります。ディスクのボトルネックを防ぐには、ディスクにアクセスするプロセス数を最小限に抑えます。ディスクにアクセスするプロセスには、データベース関数とオペレーティングシステム関数が含まれています。並列のセッションやパイプラインパーティションにもディスクアクセスが必要です。

ページングの削減

ページングは、Integration Service プロセスで特定の処理を行う際にオペレーティングシステムのメモリが不足して、メモリの代わりにローカルディスクを使用する場合に発生します。ページングとその結果生じるパフォーマンスの低下を減らすには、メモリをさらに解放をするか、または物理的メモリを増やします。システムツールを使ってページング動作を監視してください。

システムメモリを増やして対処したほうがよい状況を以下に示します。

- キャッシュに格納された、大きなルックアップを使用するセッションを実行する場合。
- 多くのパーティションで構成されるセッションを実行する場合。

メモリを解放できない場合は、システムにメモリを追加します。

プロセッサバインドの使用

マルチプロセッサ UNIX 環境では、多数のセッションを実行したときに Integration Service が大量のシステムリソースを使用する場合があります。その結果、マシン上の他のアプリケーションが十分なシステムリソースを使用できなくなる可能性があります。プロセッサバインドを使用して、Integration Service プロセスノードによるプロセッサの使用を制御できます。また、ソースデータベースとターゲットデータベースが同じマシン上にある場合は、プロセッサバインドを使用して、データベースが使用するリソースを制限します。

Sun Solaris 環境では、システム管理者は psrset コマンドを使用して、プロセッサセットの作成と管理を行うことができます。システム管理者は次に pbind コマンドを使用して Integration Service をプロセッサセットにバインドし、そのプロセッサセットが Integration Service のみを実行するように設定できます。また、Sun Solaris 環境では、psrinfo コマンドによる設定済みの各プロセッサに関する詳細の表示、および、psradm コマンドによるプロセッサのオペレーショナルステータスの変更を行うこともできます。詳細については、システム管理者に問い合わせるか、Sun Solaris のマニュアルを参照してください。

AIX 環境の場合、システム管理者は AIX 5L の Workload Manager を使って、要求のピーク時におけるシステムリソースを管理できます。Workload Manager は、リソースを割り当て、CPU、メモリ、およびディスク I/O バンド幅を管理できます。詳細については、システム管理者に問い合わせるか、AIX のマニュアルを参照してください。

第 11 章

パイプラインパーティションの使用

この章では、以下の項目について説明します。

- [パイプラインパーティションの使用の概要, 64 ページ](#)
- [パーティション化のためのソースデータベースの最適化, 66 ページ](#)
- [パーティション化のためのターゲットデータベースの最適化, 68 ページ](#)

パイプラインパーティションの使用の概要

アプリケーション、データベース、およびシステムがシングルパーティションで最高パフォーマンスを実現できるようチューニングした後で、システムが十分には活用されていないことが判明する場合があります。この時点では、セッションに 2 つ以上のパーティションを持たせるよう設定することができます。

パイプラインパーティションを使用すると、セッションのパフォーマンスを向上させることができます。パーティションまたはパーティションポイントの数を増やすと、スレッドの数が増えます。したがって、パーティションまたはパーティションポイントの数を増やすと、Integration Service のノードに対する負荷も大きくなります。Integration Service のノードに十分な CPU バンド幅がある場合は、セッションで並列にデータ行を処理すると、セッションのパフォーマンスを向上させることができます。

注: 単一ノードの Integration Service を使用して大量のデータを処理するセッションでたくさんのパーティションまたはパーティションポイントを作成すると、システムの負荷が大きくなりすぎる場合もあります。

Partitioning オプションがある場合は、次のタスクに従ってパーティションを手動で設定します。

- パーティション数を増やす。
- パイプラインの特定のポイントにあるパーティションタイプで、最もパフォーマンスの高いものを選択する。
- 複数の CPU を使用する。

パーティション数の増加

セッションのパフォーマンスを改善するために、パイプライン内のパーティション数を増やすことができます。パーティションの数を増やすことにより、Integration Service はソースへの複数の接続を作成し、ソースデータのパーティションを並列に処理することができます。

セッションがファイルソースを使用するときは、1 つまたは複数のスレッドでソースを読み込むようにセッションを設定できます。複数のスレッドでファイルソースを読み込むようセッションを設定し、セッションのパフォーマンスを向上させます。Integration Service は、ファイルソースへの同時接続を複数作成します。

セッションを作成するときに、Workflow Manager はパーティション化に対してマッピング内の各パイプラインを検査します。Integration Service がパーティション化データを処理するときに、データの一貫性を保持できれば、パイプラインに複数のパーティションを指定することができます。

パーティションをセッションに追加する場合には、以下のヒントを使用します。

- **一度に 1 つずつパーティションを追加。** 最高の監視パフォーマンスを実現するには、パーティションを一度に 1 つずつ追加し、各パーティションの追加前にセッション設定を記録しておきます。
- **DTM バッファメモリの設定。** パーティションの数を増やす際、DTM バッファサイズも増やします。セッションに n 個のパーティションが含まれる場合は、1 つのパーティションを持つセッションの値の少なくとも n 倍の値まで DTM バッファサイズを増やします。
- **Sequence Generator にキャッシュされた値の設定。** セッションに n 個のパーティションがある場合、Sequence Generator トランスフォーメーションで [キャッシュされる値の数] プロパティを使用する必要はありません。この値を 0 より大きい値に設定する場合は、1 つのパーティションを持つセッションの元の値の少なくとも n 倍にします。
- **ソースデータの均一なパーティション化。** 各パーティションで同じ行数が抽出されるように設定します。
- **セッションの実行時にシステムを監視。** CPU サイクルに空きがある場合は、パーティションを追加してパフォーマンスを改善できます。たとえば、システムに 20% のアイドル時間があるときは、CPU サイクルに余裕がある場合があります。
- **パーティション追加後のシステムを監視。** CPU 利用率が向上しない場合、I/O タイムの待機時間が延びた場合、または合計のデータトランスフォーメーション率が下落した場合には、ハードウェアまたはソフトウェアにボトルネックがあると考えられます。I/O タイムの待機時間が大幅に延びる場合には、システムをチェックしてハードウェアにボトルネックがないか確認します。それ以外の場合、データベース設定をチェックしてください。

関連項目：

- [「バッファメモリ」 \(ページ 44\)](#)

最良のパフォーマンスを得るためのパーティションタイプの選択

パイプライン上のさまざまなポイントに対してそれぞれのパーティションタイプを指定することで、セッションのパフォーマンスを向上させることができます。セッションのパフォーマンスを最適化するには、ソースデータベースおよびターゲットデータベースにデータベースパーティション化パーティションタイプを使用します。データベースのパーティション化は、Oracle および IBM DB2 ソース、そして IBM DB2 ターゲットで使用できます。ソースデータベースのパーティション化を使用する場合、Integration Service は、データベースシステムにテーブルパーティション情報を要求し、データをセッションパーティションに取り出します。ターゲットデータベースのパーティション化を使用する場合、Integration Service は対応するデータベースパーティションノードにデータをロードします。

パイプラインパーティションおよびデータベースパーティションは複数使用することができます。パフォーマンスを向上させるには、パイプラインパーティションの数がデータベースパーティションの数と一致することを確認します。サブパーティション化した Oracle ソースのパフォーマンスを向上させるには、パイプラインパーティションの数がデータベースサブパーティションの数と一致することを確認します。

パフォーマンスを向上させるには、パイプラインでの次のパーティションポイントにおけるパーティションタイプを指定します。

- **Source Qualifier トランスフォーメーション。** 複数のフラットファイルからデータを同時に読み込むには、Source Qualifier トランスフォーメーション内の各フラットファイルに対して 1 つのパーティションを指定します。デフォルトのパーティションタイプであるパススルーを適用します。

- **Filter トランスフォーメーション。**ソースファイルのサイズが変化するため、各パーティションで処理するデータの量が異なります。Filter トランスフォーメーションに 1 つのパーティションポイントを設定し、Filter トランスフォーメーションに送られる負荷を平均化するために、ラウンドロビンパーティション化を選択します。
- **Sorter トランスフォーメーション。**Sorter トランスフォーメーションと Aggregator トランスフォーメーションで重複するグループを無くすために、Sorter トランスフォーメーションで自動ハッシュキーパーティション化を使用します。これにより、Integration Service は Sorter トランスフォーメーションと Aggregator トランスフォーメーションで行が処理される前に、同じ説明を持つすべての項目が同じパーティションにグループ化されます。Aggregator トランスフォーメーションにあるデフォルトのパーティションポイントは削除できます。
- **ターゲット。**ターゲットテーブルはキー範囲によりパーティション化されるため、ターゲットにキー範囲パーティション化を指定してターゲットへのデータの書き込みを最適化します。

複数の CPU の使用

Integration Service は、パイプラインでの読み込み、トランスフォーメーション、および書き込み処理を並列に行います。PowerCenter Server は、セッション内のパイプラインの複数のパーティションを並列に処理するほか、複数のセッションを並列に実行することができます。

対称型マルチプロセッシング（SMP）プラットフォーム上では、複数の CPU を使用してセッションデータまたはデータのパーティションを並列に処理できます。すなわち真の並列処理が実現されるので、パフォーマンスが向上します。シングルプロセッサプラットフォーム上ではこれらの作業は CPU を共有するので、並列処理は存在しません。

Integration Service は、複数の CPU を使用して、複数のパーティションを含むセッションを処理することができます。使用される CPU の数は、パーティションの数、スレッドの数、利用可能な CPU の数、およびマッピングを処理するのに必要なリソースの量などの要素に依存します。

パーティション化のためのソースデータベースの最適化

パーティションを追加して、クエリーの速度を高めることも考慮できます。通常、reader 側の各パーティションで、処理対象になるデータのサブセットを表します。

次のタスクを実行して、パーティション化のためにソースデータベースを最適化します、

- **データベースのチューニング。**データベースが適切にチューニングされない場合は、パーティションを作成してもセッションが速く処理されない可能性があります。
- **並列クエリの有効化。**データベースの中には、並列クエリを有効にするために設定する必要があるオプションが備えられているものがあります。これらのオプションに関するデータベースのマニュアルを参照してください。これらのオプションがオフの場合、Integration Service では複数のパーティションの SELECT 文が連続して実行されます。
- **データを異なるテーブルスペースに分割。**各データベースでは、データを異なるテーブルスペースに分割するオプションが提供されています。データベースが許容する場合、PowerCenter SQL オーバーライド機能を使用して、1 つのパーティションからデータを抽出するクエリを指定できます。
- **ソート済みデータのグループ化。**ソースデータのパーティション化やグループ化によって、ソートされた Joiner トランスフォーメーションのパフォーマンスを向上させることができます。
- **単一のソート済みクエリの最大化。**

データベースのチューニング

データベースが適切にチューニングされない場合は、結果的にセッションを高速化できないこともあります。データベースをテストして、チューニングが正しく行われているかを確認できます。

データベースが正しくチューニングされているかを確認するには：

1. 1つのパーティションを含むパイプラインを作成します。
2. Workflow Monitor で reader スループットを測定します。
3. パーティションを追加します。
4. スループットが直線的に拡張しているか確認します。

たとえば、セッションに2つのパーティションが存在する場合、reader スループットの速さは2倍になっているはずです。スループットが直線的に拡張していない場合は、データベースのチューニングが必要になりそうです。

ソート済みデータのグループ化

ソースデータをパーティション化したりグループ化することで、ソートされた Joiner トランスフォーメーションのパフォーマンスを向上させることができます。パーティションポイントを Sorter トランスフォーメーションの前に置くことによって、各グループ内でデータを確実にグループ化しソートすることができます。

データをグループ化するには、同じキー値を持つ行を確実に同じパーティションにルーティングする必要があります。複数のパーティションに対してデータを均等にグループ化し配分するための最も効果的な方法は、自動ハッシュキーまたはキー範囲のパーティションポイントを、ソート基点より前に追加することです。

単一のソート済みクエリの最適化

データベースで単一のソート済みクエリを最適化するには、並列処理を実現する以下のチューニングオプションを検討します。

- **自動チューニングを実行する設定パラメータの確認。**たとえば、Oracle には `parallel_automatic_tuning` というパラメータがあります。
- **内部並列処理が有効になっていることを確認。**内部並列処理は、1つのクエリで複数のスレッドを実行できる機能です。たとえば、Oracle では `parallel_adaptive_multi_user` を確認してください。DB2 では `intra_parallel` を確認してください。
- **並列実行に利用可能な並列処理の最大数の確認。**たとえば、Oracle の場合は、`parallel_max_servers` を確認します。DB2 では `max_agents` を確認します。
- **並列化で使用されている各種リソースのサイズの確認。**たとえば、Oracle には、`large_pool_size`、`shared_pool_size`、`hash_area_size`、`parallel_execution_message_size`、`optimizer_percent_parallel` などのパラメータがあります。DB2 には、`dft_fetch_size`、`fcm_num_buffers`、`sort_heap` などのパラメータがあります。
- **並列度の確認。**このオプションは、データベース設定パラメータ、またはテーブルやクエリのオプションとして指定できます。たとえば、Oracle には `parallel_threads_per_cpu` および `optimizer_percent_parallel` というパラメータがあります。DB2 には、`dft_prefetch_size`、`dft_degree`、`max_query_degree` などの設定パラメータがあります。
- **データベースの拡張性に影響を及ぼす可能性のあるオプションをオフ。**たとえば、Oracle では、アーカイブリングと時間統計を無効にします。

データベースチューニングオプションのすべてのリストについては、データベースのマニュアルを参照してください。

パーティション化のためのターゲットデータベースの最適化

セッションに複数のパーティションが含まれている場合は、各パーティションのスループットをシングルパーティションセッションのスループットと同じにするとよいでしょう。この相関関係がない場合は、データベースはおそらくレコードを連続的に挿入しています。

データベースが行を並列に挿入しているかどうかを確認するには、ターゲットデータベースで以下の設定オプションをチェックしてください。

- **データベースにオプションを設定して並列挿入を有効化。**たとえば、`db_writer_processes`を設定し、Oracle データベースの DB2 の `max_agents` オプションで、並列化挿入を有効にします。データベースの中には、これらのオプションをデフォルトで有効にしているものもあります。
- **ターゲットテーブルのパーティション化の考慮。**可能な場合、Router トランスフォーメーションを使用し、各パーティションでシングルデータベースパーティションへの書き込みを行います。また、パイプラインパーティション間の I/O 競合を防ぐために、別々のディスクにデータベースパーティションを設けることを検討してください。
- **データベースにオプションを設定してデータベースの拡張性を向上。**たとえば、Oracle データベースでは、アーカイブロギングや時間統計を無効にして、拡張性を向上させます。

付録 A

パフォーマンスカウンタ

この付録では、以下の項目について説明します。

- [パフォーマンスカウンタの概要, 69 ページ](#)
- [Errorrows カウンタ, 69 ページ](#)
- [Readfromcache カウンタおよび Writetocache カウンタ, 70 ページ](#)
- [Readfromdisk および Writetodisk カウンタ, 70 ページ](#)
- [Rowsinlookupcache カウンタ, 71 ページ](#)

パフォーマンスカウンタの概要

すべてのトランスフォーメーションはカウンタを持っています。Integration Service では、トランスフォーメーションごとに入力行、出力行、およびエラー行の数が追跡されます。一部のトランスフォーメーションには、パフォーマンスカウンタを持つものがあります。以下のパフォーマンスカウンタを使用すると、セッションのパフォーマンスを向上させることができます。

- Errorrows
- Readfromcache および Writetocache
- Readfromdisk および Writetodisk
- Rowsinlookupcache

Errorrows カウンタ

トランスフォーメーションのエラーは、セッションのパフォーマンスに影響を与えます。

Transformation_errorrows カウンタのいずれかで、トランスフォーメーションに多数のエラー行がある場合は、エラーを除去するとパフォーマンスを改善できます。

関連項目：

- [「トランスフォーメーションエラーの除去」 \(ページ 42\)](#)

Readfromcache カウンタおよび Writetocache カウンタ

セッションに Aggregator、Rank または Joiner の各トランスフォーメーションが含まれる場合は、*Transformation_readfromcache* および *Transformation_writetocache* カウンタ以外にも、*Transformation_readfromdisk* および *Transformation_writetodisk* カウンタを確認して、Integration Service によるディスクからの読み出しまたはディスクへの書き込みを分析します。セッションの実行中にセッションのパフォーマンス詳細を表示するには、Workflow Monitor でセッションを右クリックし、[プロパティ] を選択します。詳細ダイアログボックスで [プロパティ] タブをクリックします。

ディスクアクセスを分析するには、最初にヒット/ミス率を計算します。ヒット率は、Integration Service がキャッシュに対して実行する読み出し/書き込み操作回数を示します。

ミス率は、Integration Service がディスクに対して実行する読み出し/書き込み操作回数を示します。

キャッシュのミス率を計算するには、次の式を使用します。

$$\frac{[(\# \text{ of reads from disk}) + (\# \text{ of writes to disk})]}{[(\# \text{ of reads from memory cache}) + (\# \text{ of writes to memory cache})]}$$

キャッシュのヒット率を計算するには、次の式を使用します。

$$[1 - \text{Cache Miss ratio}]$$

ディスクへの読み出し/書き込みを最小限に抑えるには、キャッシュサイズを増やしてください。最適なキャッシュヒット率は 1 です。

Readfromdisk および Writetodisk カウンタ

セッションに Aggregator、Rank または Joiner の各トランスフォーメーションが含まれる場合は、*Transformation_readfromdisk* および *Transformation_writetodisk* の各カウンタを確認します。セッションの実行中にセッションのパフォーマンス詳細を表示するには、Workflow Monitor でセッションを右クリックし、[プロパティ] を選択します。詳細ダイアログボックスで [プロパティ] タブをクリックします。

これらのカウンタにゼロ以外の数値が表示されている場合は、キャッシュサイズを大きくするとセッションのパフォーマンスを改善できます。Integration Service では、グループ情報の格納にはインデックスキャッシュが使用され、トランスフォーメーション後のデータの格納にはデータキャッシュが使用されます。一般に、トランスフォーメーション後のデータの方が、グループ情報よりも大きなサイズになります。したがって、インデックスキャッシュとデータキャッシュの両方のサイズがパフォーマンスに影響するものの、データキャッシュのサイズを大きくしなければならない可能性の方が、インデックスキャッシュを大きくしなければならない可能性より高くなります。ただし、データキャッシュサイズ拡張によってパフォーマンスを向上できるのは、処理済みデータのボリュームが使用可能メモリより大きい場合です。

たとえば、Integration Service ではインデックスキャッシュの格納に 100MB、データキャッシュの格納に 500MB が使用されます。200 回のアクセスをインデックスキャッシュおよびデータキャッシュのそれぞれに対してランダムに分散させる場合、次のキャッシュ設定方法があります。

- パフォーマンスの最適化を目的に、インデックスキャッシュに 100MB、データキャッシュに 200MB の割り当て。Integration Service は、データの 100%をインデックスキャッシュからアクセスし、データの 40%をデータキャッシュからアクセスします。Integration Service はインデックスキャッシュへは常時アクセスしますが、そのうち 120 回はデータキャッシュにはアクセスしません。その結果、データアクセス比率は 70%となります。
- インデックスキャッシュに 50MB、データキャッシュに 250MB の割り当て。Integration Service は、データの 50%をインデックスキャッシュからアクセスし、データの 50%をデータキャッシュからアクセスします。Integration Service は、インデックスキャッシュへのアクセスと、データキャッシュへのアクセスをそれぞれ 100 回ずつ行いますが、一度に両方にアクセスすることはありません。その結果、データアクセス比率は 50%となります。

セッションで差分集計を実行する場合は、Integration Service はセッション中にローカルディスクから履歴集計データを読み込み、履歴データの保存時にディスクに書き込みます。その結果、Aggregator_readtodisk カウンタおよび Aggregator_writetodisk カウンタには、ゼロ以外の数字が表示されます。

ただし、Integration Service ではセッション終了時にファイルに履歴データが書き込まれるため、セッション中にこれらのカウンタを評価することもできます。セッションの実行中、カウンタにゼロ以外の数値が表示される場合は、キャッシュサイズをチューニングするとパフォーマンスを向上させることができます。ただし、メモリの割り当てまたは割り当て解除にはコストが生じるため、Integration Service でのデータ処理量を把握している場合は、キャッシュサイズの拡張をなるべく控えてデータボリュームの増大に対処します。

Rowsinlookupcache カウンタ

複数のルックアップはセッションの処理を低下することがあります。セッションのパフォーマンスを改善するには、大きな Lookup テーブルに対する Lookup 式をチューニングします。

関連項目：

- [「複数のルックアップの最適化」 \(ページ 38\)](#)

索引

記号

クラスタ化されたファイルシステム
共有ファイルシステムも参照
aaa] [52](#)
高可用性 [52](#)
CPU
コンカレントワークフローのための複数 [62](#)
パイプラインのパーティション化のための複数 [66](#)
DB2
PowerCenter リポジトリのパフォーマンス [58](#)
FastExport
Teradata ソース用 [25](#)
IBM DB2
リポジトリデータベーススキーマ、最適化 [58](#)
Microsoft SQL Server
インメモリデータベース [26](#)
リポジトリデータベーススキーマ、最適化 [58](#)
ネットワークファイルシステム
共有ファイルシステムを参照
aaa] [52](#)
Oracle
IPC プロトコル [25](#)
外部ローダ [21](#)
接続の最適化 [25](#)
ターゲットのチューニング [22](#)
Sybase IQ
外部ローダ [21](#)
Windows
ボトルネック [18](#)
ボトルネック、除去 [18](#)

A

Aggregator トランスフォーメーション
限定ポート接続による最適化 [34](#)
差分集計 [34](#)
ソート済み入力による最適化 [34](#)
チューニング [33](#)
パフォーマンスの詳細 [70](#)
フィルタによる最適化 [34](#)
ポート別のグループでの最適化 [34](#)
ASCII モード
パフォーマンス [60](#)

C

Char データ型
末尾の空白の削除 [31](#)
Custom トランスフォーメーション
関数呼び出しの最小化 [35](#)
チューニング [35](#)
データブロックの処理 [35](#)

D

DECODE 関数
LOOKUP 関数との比較 [31](#)
最適化に使用 [31](#)
DTM バッファ
最適なプールサイズ [45](#)

E

External Procedure トランスフォーメーション
データのブロック [32](#)

I

IIF 式
チューニング [31](#)
Integration Service
グリッド [43](#)
コミット間隔 [47](#)
最適なデータベースドライバ [60](#)
チューニング [60](#)
IPC プロトコル
Oracle ソース [25](#)

J

Joiner トランスフォーメーション
ソート済みデータ [35](#)
チューニング [35](#)
パフォーマンスの詳細 [70](#)
マスターソースの指定 [35](#)

L

LOOKUP 関数
DECODE 関数との比較 [31](#)
最適化のための最小化 [31](#)
Lookup トランスフォーメーション
ORDER BY 文による最適化 [37](#)
インデックス処理による最適化 [38](#)
キャッシュ削減による最適化 [37](#)
キャッシュによる最適化 [36](#)
コンカレントキャッシュによる最適化 [37](#)
最適化 [71](#)
大容量メモリ搭載マシンによる最適化 [38](#)
チューニング [36](#)
データベースドライバによる最適化 [36](#)
複数のルックアップ式の最適化 [38](#)
ルックアップ条件の一致の最適化 [37](#)
ルックアップ条件の最適化 [38](#)

O

ORDER BY

Lookup トランスフォーメーションのための最適化 [37](#)

P

PowerCenter リポジトリ

DB2 のパフォーマンス [58](#)

最適な場所 [58](#)

チューニング [57](#)

R

Rank トランスフォーメーション

パフォーマンスの詳細 [70](#)

S

Sequence Generator トランスフォーメーション

グリッドパフォーマンス [55](#)

再利用可能 [39](#)

チューニング [39](#)

Single-Pass 読み込み

定義 [28](#)

Sorter トランスフォーメーション

チューニング [40](#)

パーティションディレクトリの最適化 [40](#)

メモリ割り当てによる最適化 [40](#)

Source Qualifier トランスフォーメーション

チューニング [41](#)

SQL トランスフォーメーション

チューニング [41](#)

Sybase ASE

インメモリデータベース [26](#)

T

Teradata FastExport

ソースのパフォーマンス [25](#)

U

UNIX

システムのボトルネック [18](#)

プロセッサバインド [63](#)

ボトルネック、除去 [18](#)

V

Varchar データ型

末尾の空白の削除 [31](#)

X

XML ソース

バッファメモリの割り当て [44](#)

XML トランスフォーメーション

チューニング [41](#)

XML ファイル

フラットファイルとの比較 [28](#)

あ

アイドル時間

スレッド統計 [13](#)

い

一時ファイル

最適なストレージ [51](#)

一括ロード

リレーショナルターゲットのチューニング [21](#)

インデックス

Lookup テーブル用 [38](#)

削除 [20](#)

インデックスキャッシュ

最適なサイズ [47](#)

最適な場所 [46](#)

え

永続キャッシュ

ルックアップ用 [37](#)

永続キャッシュファイル

最適なストレージ [51](#)

設定ガイドライン [53](#)

エラー

トレースレベルの最小化 [48](#)

エラートレース

トレースレベルを参照

aaa] [48](#)

演算

数値対文字列 [31](#)

演算子

関数との比較 [31](#)

お

オブジェクトクエリ

条件の整理 [58](#)

か

外部ローダー

パフォーマンス [21](#)

関数

DECODE 対 LOOKUP [31](#)

演算子との比較 [31](#)

関数の呼び出し

Custom トランスフォーメーションのための最小化 [35](#)

き

キー制約

削除 [20](#)

キャッシュ

キャッシュされる行の削減 [37](#)

最適なサイズ [47](#)

最適な場所 [46](#)

シーケンス値 [39](#)

チューニング [46](#)

リポジトリメタデータ [60](#)

キャッシュディレクトリ

共有 [46](#)

キャッシュファイル
 最適なストレージ [51](#)
共有キャッシュ
 ルックアップ用 [37](#)
共有ファイルシステム
 CPU、均衡化 [52](#)
 概要 [52](#)
 高帯域幅 [51](#)
 サーバ負荷、分散 [52](#)
 設定 [52](#)
 低帯域幅 [51](#)

く

クエリ
 リレーショナルソースのチューニング [24](#)
区切りフラットファイル
 ソース [28](#)
グリッド
 Sequence Generator のパフォーマンス、向上 [55](#)
 最適な格納場所 [50](#)
 ノードのボトルネック [52](#)
 パフォーマンス [43, 50](#)
グリッド上のセッション
 Sequence Generator のパフォーマンス、向上 [55](#)

け

結合
 データベース [35](#)

こ

高可用性
 クラスタ化されたファイルシステム [52](#)
個別選択
 ソースデータのフィルタリング [41](#)
コミット間隔
 セッションのパフォーマンス [47](#)

さ

最小化
 集計関数の呼び出し [30](#)
最適なファイルストレージ
 一時ファイル [51](#)
 ソースファイル [51](#)
 ターゲットファイル [51](#)
 パラメータファイル [51](#)
 非永続キャッシュファイル [51](#)
 ログファイル [51](#)
削除
 インデックスおよびキー制約 [20](#)
 末尾の空白スペース [31](#)
差分集計
 Aggregator トランスフォーメーションの最適化 [34](#)

し

式
 チューニング [30](#)
 評価 [32](#)
 ローカル変数への置き換え [31](#)

シーケンシャル統合
 最適なファイルストレージ [51](#)
システム
 UNIX のボトルネック、特定 [18](#)
 Windows のボトルネック、特定 [18](#)
 チューニング [61](#)
 ボトルネック、Workflow Monitor を使用した特定 [17](#)
 ボトルネック、原因 [17](#)
 ボトルネック、除去 [18](#)
実行後に発信されるメール
 パフォーマンス [49](#)
集計関数
 呼び出しの最小化 [30](#)

す

数値演算
 文字列演算との比較 [31](#)
ステージングエリア
 削除 [48](#)
スペース
 末尾、削除 [31](#)
スレッド
 アイドル時間 [13](#)
 スレッド作業時間 [13](#)
 ビジー時間 [13](#)
 ボトルネック、特定 [13](#)
 ランタイム [13](#)
スレッド統計
 ボトルネック、除去 [13](#)
 ボトルネック、特定 [13](#)

せ

セッション
 グリッド [43](#)
 コンカレント [44](#)
 チューニング [43](#)
 プッシュダウンの最適化 [44](#)
 ボトルネック、原因 [17](#)
 ボトルネック、除去 [17](#)
 ボトルネック、特定 [17](#)
セッションログファイル
 無効化 [48](#)

そ

ソース
 クエリーのチューニング [24](#)
 フィルタ [25](#)
 ボトルネック、原因 [15](#)
 ボトルネック、除去 [16](#)
 ボトルネックの特定 [15](#)
 リレーショナル、チューニング [24](#)
ソースファイル
 最適なストレージ [51](#)
 フラット対 XML [28](#)
ソート済み入力
 Aggregator トランスフォーメーションの最適化 [34](#)

た

ターゲット
 バッファメモリの割り当て [44](#)

ターゲット (続く)
 ボトルネック、原因 [14](#)
 ボトルネック、除去 [14](#)
 ボトルネックの特定 [14](#)
ターゲットファイル
 最適なストレージ [51](#)
単純マッピング
 チューニング [29](#)

ち

チェックポイント間隔
 長く設定 [21](#)
抽出
 マッピングから共通ロジック [30](#)
チューニング
 Aggregator トランスフォーメーション [33](#)
 Custom トランスフォーメーション [35](#)
 Integration Service [60](#)
 Joiner トランスフォーメーション [35](#)
 Lookup トランスフォーメーション [36](#)
 PowerCenter リポジトリ [57](#)
 Sequence Generator トランスフォーメーション [39](#)
 Sorter トランスフォーメーション [40](#)
 Source Qualifier トランスフォーメーション [41](#)
 SQL トランスフォーメーション [41](#)
 XML トランスフォーメーション [41](#)
 キャッシュ [46](#)
 式 [30](#)
 システム [61](#)
 セッション [43](#)
 トランスフォーメーション [33](#)
 ネットワーク [62](#)
 ノーマライザトランスフォーメーション [39](#)
 マッピング [27](#)
 リレーショナルソース [24](#)

て

ディスク
 アクセス、最小化 [62](#)
ディレクトリ
 共有キャッシュ [46](#)
データ移動モード
 最適 [60](#)
データ型
 Char [31](#)
 Varchar [31](#)
 変換の最適化 [30](#)
データキャッシュ
 最適なサイズ [47](#)
 最適な場所 [46](#)
 接続ポート [46](#)
データフロー
 最適化 [69](#)
 モニタリング [69](#)
データベース
 Oracle ターゲットのチューニング [22](#)
 結合 [35](#)
 ソースのチューニング [24](#)
 単一のソート済みクエリーのチューニング [67](#)
 チェックポイント間隔 [21](#)
 デッドロックの最小化 [22](#)
 ネットワークパケットのサイズ [22, 25](#)
 パーティション化のためのソースの最適化 [66](#)
 パーティション化のためのターゲットの最適化 [68](#)

データベースクエリー
 ソースのボトルネック、特定 [16](#)
データベースドライバ
 Integration Service に最適 [60](#)
デッドロック
 最小化 [22](#)
テーブルスペース
 DB2 に最適なタイプ [58](#)

と

トランスフォーメーション
 エラーの除去 [42](#)
 最適化 [69](#)
 チューニング [33](#)
トランスフォーメーションスレッド
 スレッド作業時間 [13](#)
トレースレベル
 最小化 [48](#)

ね

ネットワーク
 速度の改善 [62](#)
 チューニング [62](#)
ネットワークパケット
 長く設定 [22, 25](#)

の

ノーマライザトランスフォーメーション
 チューニング [39](#)

は

パイプライン
 データフローのモニタリング [69](#)
パイプラインのパーティション化
 最適なパーティションタイプ [65](#)
 ソースデータベースの最適化 [66](#)
 ソースデータベースのチューニング [67](#)
 ターゲットデータベースの最適化 [68](#)
 パーティションの追加 [64](#)
 パフォーマンスの最適化 [64](#)
 複数の CPU [66](#)
バインディング
 プロセッサ [63](#)
バッファ長
 最適設定 [28](#)
バッファブロックサイズ
 最適 [45](#)
バッファメモリ
 割り当て [44](#)
バッファリング
 データ [32](#)
パーティション
 追加 [64](#)
パーティションタイプ
 最適 [65](#)
パフォーマンス
 チューニング、概要 [11](#)
 フラッシュ待ち時間 [48](#)
 リアルタイムセッション [48](#)
 リポジトリデータベーススキーマ、最適化 [58](#)

パフォーマンスカウンタ

Transformation_errorrows [69](#)
Transformation_readfromcache [70](#)
Transformation_readfromdisk [70](#)
Transformation_writetocache [70](#)
Transformation_writetodisk [70](#)
Rowsinlookupcache [71](#)
タイプ [69](#)

パラメータファイル

最適なストレージ [51](#)
パフォーマンスのガイドライン [54](#)

ひ

非永続キャッシュ

最適なストレージ [51](#)

ビジー時間

スレッド統計 [13](#)

評価

式 [32](#)

ふ

ファイル共有

ネットワークファイルシステム [52](#)
クラスタファイルシステム [52](#)

ファイルシステム

共有、設定 [52](#)
クラスタ [52](#)
ネットワーク [52](#)

ファイルストレージ

共有ファイルシステム [50](#)
タイプ [50](#)
ローカル [50](#)

フィルタ

ソース [25](#)

Filter トランスフォーメーション

ソースのボトルネック、特定 [15](#)

フィルタリング

ソースデータ [41](#)
データ [29](#)

プッシュダウンの最適化

パフォーマンス [44](#)

フラッシュ待ち時間

パフォーマンス、向上 [48](#)

フラットファイル

XML ファイルとの比較 [28](#)
区切りソースファイル [28](#)
最適な格納場所 [62](#)
ソースの最適化 [27](#)
バッファ長 [28](#)

フラットファイルターゲット

セッション実行後のメール [49](#)

プロセッサ

バインディング [63](#)

へ

ページサイズ

リポジトリデータベーススキーマを最適化するための最小 [58](#)

ページング

削減 [62](#)

変換

データ型 [30](#)

ほ

ポート

接続される、制限 [46](#)

ポート別のグループ

Aggregator トランスフォーメーションの最適化 [34](#)

ボトルネック

UNIX [18](#)

Windows [18](#)

識別 [12](#)

システム [17](#)

除去 [12](#)

スレッド統計 [13](#)

セッション [17](#)

ソース [15](#)

ターゲット [14](#)

マッピング [16](#)

ま

マッピング

Single-Pass 読み込み [28](#)

共通ロジックの抽出 [30](#)

単純マッピング、チューニング [29](#)

チューニング [27](#)

ボトルネック、除去 [16](#)

ボトルネック、特定 [16](#)

め

メソッド

データのフィルタリング [29](#)

メモリ

64-bit PowerCenter [47](#)

Microsoft SQL Server データベース [26](#)

Sybase ASE データベース [26](#)

長く設定 [62](#)

バッファ [44](#)

も

文字列演算

最小化 [31](#)

数値演算との比較 [31](#)

よ

読み込みテストマッピング

ソースのボトルネック、特定 [15](#)

ら

ランタイム

スレッド統計 [13](#)

り

リアルタイムセッション

パフォーマンス、向上 [48](#)

リポジトリ

データベーススキーマ、最適化 [58](#)

リポジトリサービス
リポジトリメタデータのキャッシング [60](#)
リポジトリサービスプロセス
最適な場所 [58](#)

る

ルックアップ SQL オーバーライドオプション
キャッシュサイズの削減 [37](#)
ルックアップ条件
一致 [37](#)
最適化 [38](#)

ろ

ローカル変数
式の置き換え [31](#)
ログファイル
最適なストレージ [51](#)

わ

ワークフロー
コンカレント [44](#)
ワークフローログファイル
無効化 [48](#)