independIT Integrative Technologies GmbH
Bergstraße 6
D–86529 Schrobenhausen

schedulix!focus

Parallelisierung von Prozessen mit der schedulix Dynamic Submit Funktion

Dieter Stubler Ronald Jeninga

November 25, 2016

Parallelisierung von Prozessen mit der schedulix Dynamic Submit Funktion

Einführung

Typisch für Data Warehouse Umgebungen ist die Verarbeitung großer Datenmengen. Abhängig von den zur Verfügung stehenden Hardware Ressourcen ist früher oder später der Zeitpunkt erreicht, an dem sich ein Verarbeitungsschritt nicht mehr durch Ausführung auf einem Prozessor bzw. als ein Prozess abbilden läßt. Die Ursachen dafür sind:

- Zeitliche Anforderungen erfordern die Nutzung von mehreren Prozessoren
- Systemressourcen (Memory, Plattenplatz, Temp Tablespace, Rollback Segmente, ...) stehen nur begrenzt zur Verfügung.
- Auftretende Fehler erfordern häufige Wiederholung der Verarbeitung

Parallelisierung durch RDBMS Parallel Processing

Moderne Datenbanksysteme erlauben paralleles Query Processing. Durch den Einsatz dieses Features können Abfragen und teilweise auch Datenmanipulationen auf grosse Datenmengen datenbankintern parallelisiert und auf mehreren Prozessoren ausgeführt werden.

Vorteile dieser Lösung sind:

- Kein bzw. geringer Entwicklungsaufwand
- Overhead durch Parallelisierung gering

Nachteile dagegen sind:

- Kontrolle des Parallelitätsgrades ist nur eingeschränkt bzw. aufwändig möglich
- Anpassung der Anzahl zu einer Zeit auszuführende parallelen Teilprozesse zur Laufzeit im allgemeinen nicht möglich
- Bei Auftreten eines Fehlers geht die bereits geleistete Arbeit vollständig verloren
- Erforderliche Datenbank Systemressourcen (Temp Tablespace, Rollback Segmente, ...) müssen für die komplette Operation ausreichend zur Verfügung stehen
- In Systemen in welchen Ressourcenkontrolle sehr wichtig ist, sind datenbankintern parallelisierte Prozesse problematisch, da ihr Ressourcenverbrauch oft nicht deterministisch ist

- Einfluß der Parallelisierung auf das Restsystem ist nur schwer vorhersehbar bzw. planbar

RDBMS Parallel Processing eignet sich deshalb in erster Linie zur Beschleunigung von Operationen durch den Einsatz mehrerer Prozessoren für eine Verarbeitung. Stehen Systemressourcen nicht im Überfluss zur Verfügung machen sich die Nachteile stärker bemerkbar. Dies gilt besonders für, trotz Parallelisierung, langlaufender Verarbeitungen.

Parallelisierung auf Anwendungsebene

Alternativ zur RDBMS internen Parallelisierung ist es meist möglich eine Verarbeitung schon anwendungsseitig in parallel ausführbare Teilaufgaben zu zerlegen und diese dann parallel auszuführen.

Vorteile dieser Lösung sind:

- Die volle Kontrolle des Parallelitätsgrades ist möglich.
- Anzahl zu einer Zeit aktiven Parallelprozesse ist dynamisch regelbar.
- Fehler in einem Parallelprozess invalidieren nicht die Arbeit anderer erfolgreich gelaufener Parallelprozesse. Die Auswirkungen von Fehlern auf die Gesamtlaufzeit wird dadurch reduziert.
- Systemressourcen müssen nur für die gleichzeitig laufenden Parallelprozesse zur Verfügung stehen
- Ressourcenverbrauch ist besser planbar und der Einfluß auf das Restsystem dynamisch beeinflussbar.

Nachteile dagegen sind:

- Die Implementierung ist ohne Unterstützung eines geeigneten Scheduling Systems sehr aufwändig
- Der Overhead für Ergebniszusammenführung typischerweise ist höher als bei RDBMS interner Paralleliserung

Die Gegenüberstellung der Vor- und Nachteile der anwendungsbasierten Parallelisierung macht deutlich, dass vor allem für sehr aufwändige und lang laufenden Verarbeitungsprozesse in Umgebungen mit eingeschränkter Ressourcenverfügbarkeit die anwendungsbasierte Parallelisierung einer RDBMS internen Parallelisierung vorzuziehen ist. Gerade in Data Warehouse Umgebungen sind Verarbeitungen mit oben genannten Eigenschaften sehr häufig anzutreffen.

Implementierung

Die Implementierung von anwendungsbasierter Parallalelisierung erfordert immer folgenden 3(4) Teilaufgaben:

1. Zerlegung einer Verarbeitung in parallel verarbeitbare Teilprozesse
2. Realisierung des Teilprozesses
3. Steuerung der parallelen Ausführung der Teilprozesse
4. Optional die Zusammenführung der Teilergebnisse zu einem Gesamtergebnis

Beispiel:

In einem Data Warehouse existiert ein SQL Script welches über eine sehr große partitionierte Datenbanktabelle ein Aggregat erzeugt und dieses in einer Ergebnistabelle ablegt. Da die Verarbeitung inzwischen den zur Verfügung stehenden TEMP Bereich der Datenbank sprengt, soll diese parallelisiert werden.
Die Realisierung von 1. besteht im wesentlichen daraus, eine Liste der Partitionen zu erzeugen und für jede dieser Partitionen einen Teilprozess anzustoßen.
Für 2. ziehen wir das ursprüngliche SQL Script heran. Wir ändern dieses so, daß die Aggregation nicht auf die ganze Tabelle, sondern nur auf eine, als Parameter übergebene, Partition der Tabelle erfolgt. Das Ergebnis legen wir in einem Zwischenaggregat ab.
Nach Ausführung aller parallelen Teilprozesse muss für 4. das Zwischenaggregat nochmals aggregiert werden und in die Ergebnistabelle geschrieben werden. Auch für diese Aufgabe liefert das ursprüngliche SQL Script eine gute Vorlage.
Die obigen Realisierungsmaßnahmen sind typischerweise in wenigen Stunden realisierbar.
Das Sorgenkind der Realiserung stellt Punkt 3. dar.
Will man alle Vorteile der anwendungsbasierten Parallelisierung erzielen, so muss ein Steuerungsmechanismus implementiert werden, der mindestens folgende Funktionen bietet:

- Ausführung und Fehlererkennung der parallelen Teilprozesse
- Steuerung der Anzahl gleichzeitig laufender Parallelprozesse (zur Laufzeit)
- Überwachung und Wiederanlauf von Teilprozessen nach einem Fehler.

Muß hier individuelle Entwicklung (Scripting, ...) betrieben werden, ist erhebliches Know How erforderlich und eine effiziente, stabile und auch wartbare Lösung mit vertretbarem Kostenaufwand kaum zu erreichen. Lassen sich die parallelen Teilprozesse nicht in das eingesetzte Scheduling System einbinden, so entziehen

sich diese einer wirksamen Kontrolle und Steuerung im Betrieb des Data Warehouse Gesamtsystems.
Das schedulix Scheduling System dagegen, bietet alle Funktionen um den Punkt 3. der Implementierung (siehe oben) vollständig abzudecken. Damit sind für die Realisierung der schwierigsten Aufgabe in diesem Kontext keinerlei Entwicklungsaufwände notwendig. Das größte Hindernis für die Parallelisierung von Prozessen auf Anwendungsebene ist damit entschärft.

Implementierung im schedulix Scheduling System mit Dynamic Submits

Das schedulix Scheduling System erlaubt es einem Job über das schedulix API, hierarchisch untergeordnete Child Jobs mit unterschiedlichen Parametern zu erzeugen. Diese sind dann im Scheduling System wie alle anderen Jobs eines Ablaufes sichtbar. Alle Funktionen (Monitoring, Operation, Ressourcenmanagement, ...) des schedulix Scheduling Systems stehen uneingeschränkt auch für diese dynamisch erzeugten Job Instanzen zur Verfügung.

Zurück zum Beispiel:

Abbildung 1 zeigt die Definition des parallelisierten Ablaufes Im schedulix Scheduling System.

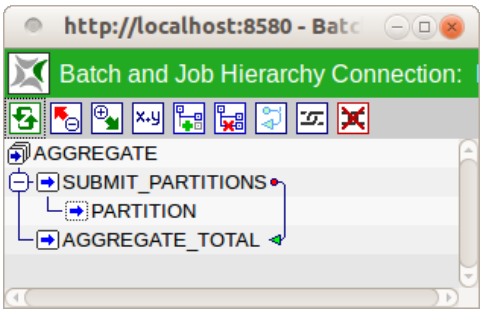

Figure 1: Definition der Aggregation im schedulix Scheduling System

Wird der Batch AGGREGATE mit einem Submit zur Ausführung gebracht, werden dabei die beiden statischen Child Jobs SUBMIT_PARTITIONS und AGGREGATE_TOTAL erzeugt und der Job SUBMIT_PARTITIONS kann anlaufen. Durch die Abhängigkeit (Pfeil im Bild) von SUBMIT_PARTITIONS wird AGGREGATE_TOTAL erst ausgeführt nachdem der SUBMIT_PARTITIONS Job inklusive aller seiner Child Jobs abgeschlossen ist.
Das dem SUBMIT_PARTITIONS Job zugrunde liegende Programm stellt die Realisierung von Punkt 1. unserer Teilaufgaben zur Parallelierung dar. Es ermittelt die Partitions der Datenbanktabelle und erzeugt über das schedulix API (z.B.: commandline command 'submit') für jede Partition einen Child Job.

Das für den Job SUBMIT_PARTITIONS auszuführende Kommando kann in einer UNIX Umgebung wie folgt definiert werden:

```
sh -c "for P in P1 P2 P3 P4 P5 P6 P7
do
    submit --host $SDMSHOST --port $SDMSPORT --jid $JOBID \\
        --key $KEY --job PARTITION --tag \$P PARTITION \$P
    if [ \$? != 0]; then exit 1; fi
done"
```

Im Monitoring des Scheduling Systems stellt sich ein laufender AGGREGATE Batch nun wie in Abbildung 2 dar.

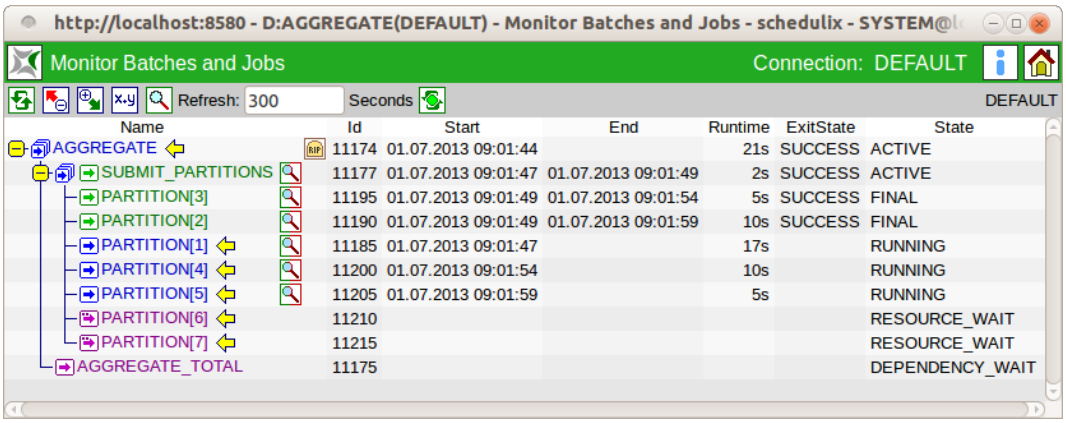

Figure 2: Monitoring Window einer laufenden Aggregation

Jeder einzelne der AGGREGATE_PARTITION Jobs ist nun über das schedulix Scheduling überwachbar und kann im Fehlerfall einzeln wiederholt (Rerun) werden.
Durch wenige Handgriffe kann durch Definition einer Ressource und Requirements für diese Ressource am Job AGGREGATE_PARTITION geregelt werden, wieviele der AGGREGATE_PARTITION Jobs zu einer Zeit vom System ausgeführt werden sollen. Diese Anzahl läßt sich auch während der Laufzeit anpassen. Abbildung 2 zeigt den Ablauf mit einer aktuellen Einstellung von maximal drei parallelen AGGREGATE_PARTITION Jobs.
Zusätzlich können die AGGREGATE_PARTITION Jobs durch Requirements für Ressourcen zur Abbildung von zur Verfügung stehenden Systemressourcen in das Ressourcen Management des übrigen Data Warehouse Betriebs eingebunden werden.

Schlußbemerkung

Das schedulix Scheduling System ermöglicht mit seiner Dynamic Submit Funktionalität eine schnelle, kostengünstige, stabile und wartbare Realisierung von anwendungsbasierter Parallelisierung.

Die parallelen Teilprozesse werden in den Ablauf eingebunden, und werden damit im Scheduling system sichtbar. Die Übersicht und Kontrolle über jeden dieser Teilprozesse ist damit zu jeder Zeit gegeben.