independIT Integrative Technologies GmbH
Bergstraße 6
D–86529 Schrobenhausen

schedulix!focus

Job Scheduling als Alternative zur Ablaufsteuerung mit Shell Scripten

Dieter Stubler Ronald Jeninga

November 25, 2016

Enterprise Job Scheduling als Alternative zur Ablaufsteuerung mit Shell Scripten

Vorbemerkung

Diesem Dokument beschäftigt sich mit den Problemen des Einsatzes der UNIX Shell (sh, ksh, bash, ...) zur Ablaufsteuerung von (Teil)Prozessen. Die auftretenden und in diesem Dokument beschriebenen Schwierigkeiten sind jedoch konzeptioneller Art und gelten damit auch für den Einsatz anderer Scriptsprachen (perl, python, ...).
Keineswegs sollen die erwähnten Sprachen pauschal kritisiert werden. Jede der genannten Sprachen hat ihre Vorteile und die Autoren haben selbst in ihrer Laufbahn schon viele erfolgreiche Projekte mit dem Einsatz solcher Scriptsprachen abgewickelt.

Ablaufsteuerung von (Teil)Prozessen

In jeder IT Abteilung existieren Aufgaben, welche durch die Abarbeitung von mehreren einzelnen Prozessen erfüllt werden müssen. Solche Programmabläufe bestehen aus einigen wenigen bis mehreren hundert einzelnen Prozessen.
Um solche Abläufe abzuarbeiten, muß sichergestellt werden, dass die einzelnen Prozesse koordiniert, synchronisiert und in der richtigen Reihenfolge abgearbeitet werden. Dazu ist eine Ablaufsteuerung nötig.
In der Praxis werden häufig die UNIX Shell oder andere Scriptsprachen eingesetzt, um solche Abläufe inklusive deren Steuerung zu implementieren.

Zu Beginn ist alles einfach

In der Startphase eines Projektes ist die Welt immer ganz einfach. Die Anzahl von Abläufen ist klein, deren Komplexität gering, die Datenmengen überschaubar und Ablaufsteuerung ist (noch) kein Thema. Mit ein paar Zeilen Shell Script ist so ziemlich jeder Ablauf geregelt, schnell, pragmatisch und kostengünstig.
Genau dies birgt jedoch ein großes Risiko, da diese Lösungen mit steigender Komplexität und erhöhten Anforderungen an die Performance und Zuverlässigkeit des Gesamtsystems erhebliche Entwicklungsressourcen binden und mittelfristig einen zuverlässigen Betrieb der Abläufe zu angemessenen Kosten unmöglich machen.
Aus Sicht des Managements ist dabei fatal, dass die Kosten für Ablaufsteuerung und Überwachung anfangs so gering sind, dass diese Kosten nicht einmal gesondert budgetiert werden. Das immer auftretende schleichende Anwachsen dieser Kosten bleibt der Führung deshalb sehr lange verborgen (Blindleistung!). Wird das Thema als Problem erkannt, sind bereits enorme Summen in der Entwicklung von Steuerscripten und der dazu nötigen Hilfsroutinen und Frameworks versickert.
Um diese Investitionen nicht zu verlieren und aus Furcht vor den Kosten einer Umstellung der Ablaufsteuerung auf ein geeignetes Job Scheduling System wird

daher oft weiter versucht, an den bestehenden Arbeitsweisen festgehalten. Die Kosten steigen weiter, immer mehr Ressourcen der Mitarbeiter werden gebunden und eine Abkehr von dieser ineffizienten Arbeitsweise wird immer teurer.
Wir wollen in diesem Dokument für diese Problematik sensibilisieren und deutlich machen, dass ein frühzeitiger Einsatz eines geeigneten Job Scheduling Systems das Entstehen des beschriebenen Dilemmas vermeidet und dabei von Anfang an Kosten reduziert und Ressourcen für die Lösung der eigentlichen Aufgaben freisetzt.

Ein minimales Beispiel

Wir wollen die Entwicklung eines einfachen Ablaufes an einem ganz simplen Beispiel exemplarisch darstellen. Dazu gehen wir davon aus, dass für einen einfachen Ablauf zwei Programme (P1 und P2) nacheinander ausgeführt werden sollen.
Wir erstellen dazu folgendes sh Script:

```
#!/bin/sh
P1
P2
```

Das ist nun wirklich einfach, oder ?

Fehlerbehandlung

Fehler werden in obigen Beispiel nicht behandelt. Damit nun P2 nicht auf falschen Daten arbeitet, muß verhindert werden, dass P2 anläuft, falls P1 einen Fehler liefert. Nachdem dies nun schon mehrmals zu Ärger geführt hat, müssen wir eine Fehlerbehandlung in unser Script einbauen. Das sieht dann so aus:

```
#!/bin/sh
P1
RET=$?
if [ $RET -ne 0 ]
then
        echo "Fehler $RET in P1 !"
        exit $RET
fi
P2
RET=$?
if [ $RET -ne 0 ]
then
        echo "Fehler $RET in P2 !"
        exit $RET
fi
```

Wie wir sehen, sind da schon ein paar Zeilen hinzugekommen. Diese müssen noch getestet werden, da ein nicht behandelter Fehler fatale Folgen haben kann. Schon die Fehlerbehandlung wird von unterschiedlichen Entwicklern leicht unterschiedlich implementiert werden. Verständlichkeit und Wartbarkeit sinken und die Kosten steigen.

Wiederanlauf

In unserem Beispiel hat P1 eine Laufzeit von 3 Stunden und P2 liefert immer wieder einen Fehler durch einen Ressourcenengpass in einem temporären Speicherbereich. Ein Neustart des Scripts würde auch überflüssigerweise auch P1 erneut ausführen, was zu einer 3 Stunden höheren Reparaturzeit führt.
In vielen Umgebungen wird der Entwickler nun das Script kopieren, den Aufruf von P1 auskommentieren und die Kopie des Scripts ausführen. Dies verursacht eine Menge Arbeit und birgt ein hohes Fehlerrisiko.
Da dies nicht akzeptabel ist, benötigt das Script etwas Gedächtnis und sieht dann so aus:

```
#!/bin/sh
STEP=`cat stepfile`
if [ $STEP -eq "0" ]
then
        P1
        RET=$?
        if [ $RET -ne 0 ]
        then
                echo "Fehler $RET in P1 !"
                exit $RET
        fi
        echo "1" > stepfile
fi
if [ $STEP -eq "1" ]
then
        P2
        RET=$?
        if [ $RET -ne 0 ]
        then
                echo "Fehler $RET in P2 !"
                exit $RET
        fi
        echo "2" > stepfile
fi
echo "0" > stepfile
```

Dies ist eine schnelle Möglichkeit, für die sich der Entwickler entscheiden könnte. Obige Lösung birgt jedoch eine Menge Probleme. Das ‘stepfile’ muß vor der ersten Ausführung und evtl. nach einem Abbruch der Verarbeitung, initialisiert werden. Die Fehlerbehandlung für Lesen und Schreiben des Fortschritts fehlt noch vollständig. Mit dieser Lösung kann das Script immer nur einmal zu einer Zeit ausgeführt werden. Um das Script produktiv einsetzbar zu machen, ist daher noch erheblich mehr Entwicklungsaufwand zu investieren.
An dieser Stelle verzichten wir auch auf weitere Codebeispiele für unser Beispiel, da hier schnell mehrere hundert Zeilen Code erforderlich werden, deren Entwicklungsaufwand wir uns natürlich sparen.

Was noch ?

Das obige Beispiel beschäftigte sich nur mit den ersten, einfachen Problemen der Ablaufsteuerung per Script. Um einen stabilen, reibungslosen Betrieb sicherzustellen sind unter anderem noch folgende weitere Funktionen notwendig:

- Überwachung und Operatoreingriff
- Übergabe von Steuerinformation
- Ermöglichung von Parallelität
- Verteilte Ausführung
- Ressourcenkontrolle

Diese und weitere Funktionen in der Ablaufsteuerung zu implementieren, erfordert teilweise erheblichen Entwicklungs- und Wartungsaufwand.
Fehlen diese Funktionen, wird dies durch erhöhte Betriebskosten bezahlt werden.

Überwachung und Operatoreingriff

Um einen stabilen Betrieb sicherstellen zu können, muß es möglich sein, alle Abläufe zu überwachen. Dazu müssen alle Abläufe ihren Fortschritt protokollieren ('stepfile' im Beispiel). Diese Protokolle müssen eingesammelt und aufbereitet werden, um einen Statusüberblick über die laufenden Abläufe zu erhalten. Dazu wird ein Repository benötigt, welches Zustandsinformationen über laufende Abläufe enthält. Um über Probleme schnell informiert zu sein, ist nebenbei auch noch ein Notification System zu entwickeln, welches per Email, SMS oder andere Kanäle Meldungen verschicken kann. Allein die Entwicklung dieser Funktionen stellt ein nicht zu unterschätzendes Projekt dar, in dem schnell eine zweistellige Anzahl Manntage zu investieren sind.
Treten Fehler oder Probleme auf, muß der Operator in der Lage sein, in laufende Abläufe einzugreifen. Er muß Abläufe oder Prozesse neu starten, anhalten, abbrechen, überspringen, usw. können. Soll dies einigermassen komfortabel erfolgen, ist auch hierfür ein enormer Entwicklungsaufwand zu leisen.

Übergabe von Steuerinformation

Oft ist es notwendig, Informationen (Zeitstempel, Dateinamen, ...) von einem Prozess an einen nachfolgenden Prozess zu übergeben. In einer Script basierten Ablaufsteuerung wird dies typischerweise über Dateien implementiert, welche von einem Prozess geschrieben und von dessen Nachfolger gelesen werden. Diese Lösung birgt jedoch wiederum viele Nachteile. Spätestens wenn einer der beteiligten Prozesse auf einer anderen Maschine im Netzwerk ausgeführt werden soll, sind wieder erhebliche Entwicklungsleistungen notwendig, um die Informationsübergabe zu reimplementieren.

Ermöglichung von Parallelität

Können Teilprozesse eines Ablaufes parallel ausgeführt werden, so bekommt die Scriptentwicklung eine neue Dimension. Prozesse müssen 'im Hintergrund' gestartet werden und an bestimmten Stellen des Ablaufes muß auf die Beendigung mehrerer parallel ablaufender Prozesse gewartet werden. Dies erfordert schon ein hohes technisches Know How des Entwicklers und ist inklusive Fehlerbehandlung, Wiederanlauf, Monitoring und Operation wahrlich keine triviale Aufgabe.

Verteilte Ausführung

Sollen die Teilprozesse eines Ablaufes auf unterschiedlichen Rechnern zur Ausführung gebracht werden, so müssen nun aus den Scripten Prozesse auf anderen Rechnern gestartet werden. Auch dies ist keine einfache Aufgabe, da die Tücken von remote Kommandos (rsh, rcp, ...) bzgl. Überwachung und Fehlererkennung überwunden werden müssen.
Auf die potentiellen Sicherheitsrisiken, etwa durch das Kodieren von Passwörtern, wollen wir an dieser Stelle gar nicht eingehen.

Ressourcenkontrolle

Die zur Verfügung stehenden Systemressourcen sind immer begrenzt. Werden zu einer Zeit zu viele Ressourcen benötigt, kommt es zu Durchsatzeinbußen und einem gehäufen Auftreten von Fehlern durch Ressourcenengpässe. Es muß also sichergestellt werden, dass Prozesse erst starten, wenn genügend Ressourcen zur Verfügung stehen. Dies zu realisieren, ist mit einem vertretbaren Aufwand in einer Script basierten Ablaufsteuerung nahezu unmöglich.

Wir haben bisher nur einige der wichtigsten Aspekte der Ablaufsteuerung und der durch Scripting entstehenden Aufwände beleuchtet. Ohne den Einsatz eines geeigneten Job Scheduling Systems entstehen erhebliche laufende Kosten im Betrieb. Versucht man, diese Kosten durch Verbesserung der Script Infrastruktur zu verbessern, ist ein hoher Entwicklungsaufwand mit dem damit verbundenen Wartungsaufwand zu erbringen.

schedulix als Alternative und Ausweg

independIT bietet mit ihrem schedulix Enterprise Job Scheduling System eine Alternative und einen Ausweg aus der Script Falle.
Das schedulix Job Scheduling System bietet alle notwendigen Funktionen, um auch umfangreiche und komplexe Abläufe zu modellieren, ohne Teile der Ablaufsteuerung in den Teilprozessen selbst implementieren zu müssen.
Die Aufwände für Entwicklung, Wartung und Betrieb werden durch den Einsatz von schedulix drastisch reduziert.

Zusätzlich wird der Betrieb stabiler, weniger fehleranfällig und sicherer. Recovery Zeiten können deutlich verkürzt werden.

Schlußbemerkung

Die Steuerung von Abläufen auf Basis von Scripts erfordert in Entwicklung, Wartung und Betrieb auf Dauer unangemessen hohe Aufwände. Ein optimaler, transparenter und effizienter Betrieb ist durch eine Script basierte Steuerung der Abläufe nicht zu erreichen.
Wir empfehlen deshalb den Einsatz eines geeigneten Job Scheduling Systems schon in einer sehr frühen Phase. Je früher in laufenden Projekten ein Umstieg auf ein solches System erfolgt, desto weniger Investitionen gehen verloren und desto geringer sind die Umstellungsaufwände. Sollten Scripts zur Ablaufsteuerung im Einsatz sein, ist ein möglichst schneller Umstieg auf eine Job Scheduling System basierte Lösung anzustreben.
Das schedulix Enterprise Scheduling System bietet alle Funktionen zu Entwicklung, Betrieb, Monitoring und Operations komplexer Abläufe und minimiert damit die Kosten für Entwicklung und Wartung von Prozessabläufen. Zu einem Bruchteil der Kosten einer Scripting Lösung unterstützt schedulix den Aufbau und Betrieb eines dauerhaft stabilen und zuverlässigen IT Betriebs.