

Workaround zur Begrenzung der Startanzahl einer Applikation in Tivoli System Automation

Armin M. Warda, Bonn, 23.01.2007

Einleitung und Zusammenfassung

Diese Anleitung beschreibt einen Workaround für ein Defizit bzgl. des Restart-Verhaltens von Ressourcen in *Tivoli System Automation*. (Das Defizit existiert mindestens in allen *TSA* Versionen bis einschließlich 2.2 Fixpack 1).

Wünschenswert ist, daß eine Resource der Klasse *IBM.Application* nur eine begrenzte Zahl von Starts auf demselben Knoten innerhalb eines gegebenen Zeitintervalls erfahren soll. Falls der Start dieser Resource häufiger innerhalb dieses Zeitintervalls angefordert wird, so soll kein weiterer Start auf demselben Knoten erfolgen, sondern die Resource auf einen anderen Knoten verlagert werden, falls ein geeigneter Kandidat existiert. Ansonsten soll die Resource nicht mehr gestartet werden, bspw. falls es sich nicht um eine *floating* sondern um eine *fixed* Resource handelt, oder kein anderer Knoten die Startbedingungen der Resource erfüllt.

Leider unterstützt *TSA* das gewünschte Verhalten nur in dem Fall, wenn der Start der Resource *unmittelbar* fehlschlägt, aber nicht in dem Fall, wenn die Resource zunächst (scheinbar) erfolgreich gestartet werden kann und dann stets kurze Zeit später ausfällt.

Der Workaround besteht darin, in der Start-Methode einer Resource der Klasse *IBM.Application* die Zeitstempel der letzten Startzeitpunkte aufzuzeichnen und dahingehend zu überprüfen, ob die spezifizierte Anzahl der maximal zulässigen Starts in dem gegebenem Zeitintervall überschritten ist. Ist dies der Fall, so wird der Start unmittelbar abgebrochen, der Status der Resource für diesen Knoten auf *FAILED_OFFLINE* gesetzt, dadurch nicht mehr versucht die Resource auf diesem Knoten zu starten und somit das gewünschte Verhalten von *TSA* erreicht.

Der Workaround läßt sich in generischer Form in eine einzige Shell-Funktion mit drei Parametern kapseln, sodaß bestehende Startskripte von Ressourcen der Klasse *IBM.Application* nur um einen einzigen Aufruf dieser Funktion ergänzt werden müssen, um den Workaround zu benutzen. Dadurch wird vermieden, daß sich die Komplexität der Start-Methoden signifikant erhöht. Da der Workaround eine Modifikation der Start-Methode der Resource erfordert, kann er nur auf Ressourcen der Klasse *IBM.Application* angewendet werden, da hier die Start-Methode in Form eines Startskripts vom Systemintegrator erstellt wird. Auf Ressourcen anderer Klassen, bspw. *IBM.ServiceIP*, läßt sich der Workaround nicht anwenden, weil diese fest einprogrammierte Start-Methoden haben. Hier könnte nur der Hersteller *IBM* einen entsprechenden Workaround implementieren. Eine weitere Einschränkung des Workarounds besteht darin, daß die Spezifikation der Parameter – der maximal zulässigen Startanzahl und des Zeitintervalls – nicht wie üblich in Form von Klassen-Attributen der Resource erfolgen kann, sondern innerhalb des Startskripts erfolgt. Diese Einschränkung könnte nur entfallen, wenn der Hersteller *IBM* den Workaround integrieren würde.

Szenario

Eine Resource, die sich im Zustand *OFFLINE* befindet soll von *TSA* in den Zustand *ONLINE* gebracht, also gestartet werden.

1. *TSA* ruft die Start-Methode der Resource auf.
2. Der Start verläuft zunächst erfolgreich, was über den Return-Code *OK=0* der Start-Methode an *TSA* kommuniziert wird.

3. *TSA* ruft periodisch die Monitor-Methode der Resource auf um zu prüfen, ob die Resource im Zustand ONLINE ist.
4. Die Resource bleibt nun zunächst einige Zeit ONLINE, was über den Return-Code ONLINE=1 der Status-Methode an *TSA* kommuniziert wird.
5. Nach einiger Zeit fällt die Resource aber aus, was über den Return-Code OFFLINE=2 der Status-Methode an *TSA* kommuniziert wird.
6. Da die Resource nun im Zustand OFFLINE beobachtet wird, aber im Zustand ONLINE sein soll, wird die Resource restartet.

Die Schritte 2 bis 6 können sich nun unbegrenzt wiederholen.

Beobachtetes Verhalten

Eine Resource, die immer wieder für kurze Zeit erfolgreich gestartet werden kann, aber stets nach kurzer Zeit ausfällt, wird immer wieder auf demselben Cluster-Knoten gestartet, statt zu versuchen, sie auf einem anderen Knoten zu starten, oder sie gestoppt zu lassen. Der Zustand der Resource auf diesem Knoten wechselt dann kontinuierlich zwischen ONLINE und OFFLINE.

Gewünschtes Verhalten

Die Resource soll nach einer begrenzten Anzahl von Starts innerhalb eines definierten Zeitraums nicht mehr auf demselben Knoten gestartet werden. Stattdessen soll der Zustand der Resource auf FAILED_OFFLINE gesetzt werden und ggf. (im Falle einer *floating resource*) versucht werden, die Resource auf einem anderen Knoten zu starten, falls ein geeigneter Kandidat existiert, oder andernfalls gar nicht mehr gestartet werden.

Praxisrelevanz

Folgende Beispiele aus dem Rechenzentrumsbetrieb veranschaulichen die Relevanz des gewünschten Verhaltens:

- Eine Dialoganwendung lasse sich zunächst starten und laufe auch stabil, solange sich keine oder nur wenige Anwender anmelden. Sie stürze aber ab, sobald die Anzahl der angemeldeten Anwender eine kurze Zeit später eine bestimmte Anzahl übersteigt.
- Ein Datenbanksystem läßt sich zunächst starten und benutzen, nach einiger Zeit stürzt das System aber bei bestimmten Zugriffen ab.

Falls die Abstürze der Anwendung oder des Datenbanksystems durch Ressourcenknappheit oder Fehlkonfiguration des Betriebssystems verursacht werden, so kann das Problem i.a. nicht durch einen lokalen Restart behoben werden, möglicherweise jedoch durch eine Verlagerung der Anwendung auf einen anderen Knoten, wo diese Problemursachen nicht vorliegen müssen.

Falls hingegen bspw. eine Datenkorruption den Absturz des Datenbanksystems auslöst, so wird weder ein lokaler Restart noch eine Verlagerung des Datenbanksystems auf einen anderen Knoten die Problemursache beseitigen.

In beiden Fällen macht es jedenfalls keinen Sinn, die Anwendung oder Datenbank unbegrenzt häufig immer wieder auf demselben Knoten nur kurzzeitig erfolgreich zu starten. Tatsächlich birgt jeder weitere Startversuch und anschließender Absturz sogar die Gefahr für eine Fehlerfortpflanzung oder Verbreiterung der Datenkorruption, so daß es ratsam ist, hier eine Begrenzung vorzunehmen.

Implementierung des Workarounds

Das folgende Shell-Skript implementiert den beschriebenen Workaround. Es ist von jedem Startskript einzubinden, das den Workaround verwenden soll:

```
1 # Armin <dot> Warda <at> koeln <dot> de
2
3 . /etc/profile
4
5 # TSA codes for status-method of resources:
6     UNKNOWN=0
7     ONLINE=1
8     OFFLINE=2
9     FAILED_OFFLINE=3
10    STUCK_ONLINE=4
11    PENDING_ONLINE=5
12    PENDING_OFFLINE=6
13    MIXED=7
14
15 tsa_state_to_text()
16 {
17     case $1 in
18         $UNKNOWN)          echo UNKNOWN ;;
19         $ONLINE)           echo ONLINE ;;
20         $OFFLINE)         echo OFFLINE ;;
21         $FAILED_OFFLINE)  echo FAILED_OFFLINE ;;
22         $STUCK_ONLINE)    echo STUCK_ONLINE ;;
23         $PENDING_ONLINE)  echo PENDING_ONLINE ;;
24         $PENDING_OFFLINE) echo PENDING_OFFLINE ;;
25         $MIXED)           echo MIXED ;;
26         *)                echo UNKNOWN ;;
27     esac
28 }
29
30 tsa_logger()
31 {
32     logger -p local3.info -t TSA $*
33 }
34
35 check_restarts ()
36 {
37     NAME=$1
38     N=$2
39     T=$3
40
41     TIMESTAMPS=/var/run/${NAME}.timestamps
42     [ -e $TIMESTAMPS ] || touch $TIMESTAMPS
43     . $TIMESTAMPS # read timestamps of last restarts
44
45     Tn=$(date +%s) # seconds since 1.1.1970
46     (( DELTA = Tn - T1 ))
47     if [ $DELTA -le $T ]; then
48         tsa_logger "too many restarts of ${NAME}
49                 ($N in $DELTA < $T seconds)"
50     else
51         >$TIMESTAMPS
52         i=2; j=1; while [ $i -lt $N ]; do
53             eval typeset -i X=\T${i}
54             echo "T${j}=$X" >>$TIMESTAMPS
55             (( i=i+1 ))
56             (( j=j+1 ))
57         done
58         echo "T${j}=$Tn" >>$TIMESTAMPS
59         return 0
60     fi
61 }
```

Die Funktion verwendet eine Datei, um die Zeitstempel seiner letzten Aufrufe an seine nächsten Aufrufe zu kommunizieren. Zeile 41-43 definiert, initialisiert und liest die Datei mit den Zeitstempeln. Falls die Datei von einem vorherigen Aufruf bereits existiert, so werden die Variablen

```
T1=<Zeitstempel vom viertletzten Start>
T2=<Zeitstempel vom drittletzten Start>
T3=<Zeitstempel vom vorletzten Start>
T4=<Zeitstempel vom letzten Start>
```

durch das Einlesen der Datei definiert, falls die zulässige Anzahl Starts $N=5$ beträgt. Für andere Werte von N werden die entsprechenden Zeitstempel T_1, T_2, \dots, T_{N-1} aufgezeichnet.

In Zeile 45 werden die Sekunden seit dem 1.1.1970 ermittelt, da diese als Einheit für die Zeitstempel benutzt werden, um in Zeile 46 einfach Zeitdifferenzen berechnen zu können.

In Zeile 51 wird die Datei mit den Zeitstempeln geleert und in den Zeilen 52-58 neu gefüllt, um die Zeitstempel an den nächsten Aufruf der Funktion zu übergeben. Dabei rücken die Zeitstempel um eine Position nach unten, der alte Zeitstempel T_1 wird also verworfen, und der aktuelle Zeitstempel angehängt, sodaß für den nächsten Aufruf der Funktion die einzulesende Datei mit folgendem Inhalt erzeugt wird (wieder für $N=5$ dargestellt.):

```
T1=<Zeitstempel vom drittletzten Start>
T2=<Zeitstempel vom vorletzten Start>
T3=<Zeitstempel vom letzten Start>
T4=<Zeitstempel von diesem Start>
```

Verwendung des Workarounds

Zunächst muß die Datei mit der Funktionsdefinition eingebunden werden.

Die Funktion `check_restarts` muß mit drei Parametern aufgerufen werden:

```
check_restarts $NAME $N $T
```

Die Parameter haben folgende Bedeutung und Wertebereiche:

1. Name = eindeutige Bezeichnung (nur in Dateinamen zulässige Zeichen erlaubt)
2. N = maximale Startanzahl (eine Integerzahl größer Null)
3. T = Länge des Zeitintervalls in Sekunden (eine Integerzahl größer Null)

Der Name muß eindeutig sein und darf nur für Dateinamen zulässige Zeichen enthalten und die maximal zulässige Länge von Dateinamen nicht überschreiten, weil er auch für die Timestamp-Datei benutzt wird.

Der Return-Code der Funktion ist

- 0 = OK, falls die Anzahl von Starts im Zeitintervall noch nicht überschritten ist und ein weiterer Start erfolgen darf
- 1 = nicht OK, falls die Anzahl von Starts im Zeitintervall überschritten ist und kein weiterer Start erfolgen soll

Die Funktion kann am einfachsten eingebunden werden, indem man sie mit Parametern und dem logischen Und-Operator der Shell „&&“ vor das eigentlich Startkommando stellt:

```
check_restarts $NAME $N $T && start_the_resource ...
RC=$?
```

Der Returncode `$RC` ist 1, falls die zulässige Anzahl der Starts im Zeitintervall überschritten ist, oder er ist gleich dem Returncode des eigentlichen Startkommandos. Ein Returncode ungleich Null wird von `TSA` bei einer Startmethode als unmittelbarer Fehlschlag interpretiert und somit die Resource für diesen Knoten auf `FAILED_OFFLINE` gesetzt, was zur Folge hat, daß kein weiterer Startversuch auf diesem Knoten mehr erfolgt, bis dieser Ressourcen-Status mittels `TSA`-Kommando `resetrsrc` zurückgesetzt wird.

Beispiel

Das folgende Skript ist ein Beispiel für die Implementierung von Start-/Stop-/ und Monitor-Methoden in einem gemeinsamen Skript, das den Workaround verwendet:

```
1  #! /bin/sh
2
3  # Armin M. Warda <Armin DOT Warda AT Koeln DOT De>
4
5  . /root/tsa/common
6
7  NAME=$(basename $0)
8  N=3 # maximum 3 restarts...
9  T=60 # ...in 60 seconds
10
11 case $1 in
12     start) check_restarts $NAME $N $T && start_server
13             RC=$?
14             ;;
15     stop)  if [ "$SA_RESET" = 1 ]; then
16             stop_server --force
17             else
18             stop_server
19             fi
20             RC=$?
21             ;;
22     status) if [ -S /var/run/server_socket ]; then # socket exists
23             RC=$ONLINE
24             else
25             RC=$OFFLINE
26             fi
27             ;;
28     *)     echo "usage: $0 {start|stop|status}" >&2
29             RC=$UNKNOWN
30             ;;
31 esac
32
33 exit $RC
```

Die notwendigen Ergänzungen für die Verwendung des Workarounds sind hervorgehoben dargestellt. In Zeile 5 wird (u.a.) die Definition der Shell-Funktion `check_restarts` eingelesen. In Zeile 7-9 werden die drei Parameter für den Funktionsaufruf `check_restarts` definiert. Nach drei Starts innerhalb eines Zeitraums von einer Minute soll die Resource nicht mehr auf demselben Knoten gestartet werden. In Zeile 12 ist schließlich der Funktionsaufruf dem eigentlichen Startkommando mit dem logischen Und-Operator der Shell „&&“ vorangestellt.

Der Returncode `$RC` der Startmethode ist 1, falls die zulässige Anzahl der Starts im Zeitintervall überschritten ist, oder er ist gleich dem Returncode des eigentlichen Startkommandos. Ein Returncode ungleich Null wird von *TSA* bei einer Startmethode als unmittelbarer Fehlschlag interpretiert und somit die Resource für diesen Knoten auf `FAILED_OFFLINE` gesetzt, was zur Folge hat, daß keine weiteren Startversuche auf diesem Knoten mehr erfolgen, bis dieser Ressourcen-Status mittels *TSA*-Kommando `resetrsrc` Zurückgesetzt wird.

Warnung

Die hier vereinfacht dargestellte Implementierung und Verwendung des Workarounds enthält keine Überprüfungen auf fehlerfreie Verwendung. Bspw. darf die Funktion für dieselbe Resource immer nur mit demselben Wert für `N` aufgerufen werden, und der Name muß eindeutig und ein gültiger Dateiname sein, aber keine existierende Datei. Diese Bedingungen werden jedoch nicht überprüft und generell werden die übergebenen Parameter nicht auf Gültigkeit überprüft.