# Verteilte und hochverfügbare Anwendungen mit Erlang

Dr. Ralf Th. Pietsch

Mai 2019

Verteilte Anwendungen sind heutzutage fast schon ein Muß um den gestiegenen Bedarf an Verfügbarkeit zu realisieren. Auf der anderen Seite stellt die Entwicklung von verteilten Anwendungen auch heute noch oft eine Herausforderung für Entwickler dar.

Mit dem richtigen Ansatz und vor allem einer Programmiersprache die Nebenläufigkeit unterstützt, wird die Entwicklung einer solchen Anwendung sehr vereinfacht. Erlang unterstützt mit seinem Prozessmodell und der Standard-Library „OTP“ die Erstellung von verteilten Anwendungen, auch über Rechnergrenzen hinweg.

## Threads und „shared memory“

Um eine Anwendung horizontal – auch über mehrere Prozessoren oder Rechner – zu skalieren ist es sinnvoll den Speicher der Programmeinheiten komplett zu isolieren. Die meisten Sprachen (Java, C#, Go)[1], die heute für nebenläufige Programme eingesetzt werden, verwenden jedoch „shared memory“ zum Kommunikation der Programmeinheiten.

Kommunizieren Programmeinheiten über „shared memory“ miteinander, so muß technisch sichergestellt werden, dass jeweils nur ein Prozess Zugriff auf den geteilten Speicherbereich hat. Zur Zugriffskontrolle werden hierzu Mutex, Semaphoren und andere technische Lösungen[2] eingesetzt. Die Zugriffssteuerung verursacht einen Overhead durch die Kontrolle auf diesen Speicherbereich und bremst somit das Gesamtsystem aus; parallele Threads blockieren sich gegenseitig in der Ausführung beim Zugriff auf den „shared memory“-Bereich Kritischer als die Zugriffsteuerung ist jedoch, dass der Zustand des „shared memories“ nach dem Absturz einer Programmeinheit nicht klar ist. Das System kann nicht feststellen, ob der abgestürzte Prozess seine Daten komplett in den „shared memory“ übertragen hat oder nicht. Da nicht klar ist, ob die Daten im shared memory korrupt sind, muß vom Worst-Case ausgegangen werden. Damit wird es zumeist notwendig das Gesamtsytem neu zu starten und damit insbesondere auch die gar nicht betroffenen Threads.

Da „Shared memory“-basierte Anwendungen einen gemeinsamen Speicherbereich vorraussetzen lassen sich auch nicht ohne weiteres auf mehrere Rechner verteilen; sie skalieren damit nur bis zur Prozessorgrenze. Reicht ein Prozessor nicht mehr aus muß die Anwendung angepasst werden.

## Prozesse und Messages

Isoliert man hingegen den Speicherbereich der Programmeinheiten strikt voneinander und läßt die Programmeinheiten mittels Nachrichten kommunizieren, so entfallen alle oben genannten Nachteile. Erlang setzt genau an dieser Stelle an. Jede Programmeinheit in Erlang (Erlang-Prozess), besitzt seinen eigenen isolierten Speicherbereich, der von keinem anderen Erlang-Prozesss aus sichtbar und zugreifbar ist. Um den Speicher getrennt zu halten kommunizieren Erlang-Prozesse über Nachrichten („messages“) miteinander. Stürzt ein Erlang-Prozess aufgrund einer Fehlersituation ab, so kann er vom Erlang-Scheduler problemlos komplett gelöscht werden. Kein anderer Erlang-Prozess ist von diesem Absturz betroffen, da die Speicherbereiche exklusiv sind. Der Fehler bleibt also auf einen definierten Bereich beschränkt und hat keine Auswirkung auf andere Systemteile.

## Supervisor-Tree

Um sicherzustellen, dass abgestürtzte Prozesse neugestartet werden und ihre Arbeit fortsetzen können, bietet die Erlang-Standard-Library „OTP“[3] das sogenannte Supervisor-Worker-Model an. Hierzu wird in

[1] Dazu zählen aber auch: C++, Python, Ruby, JavaScript, etc.

[2] Weitere Stichworte sind: Monitor, Futures, Locks, synchronized, Threads, Thread-safety.

[3] „Open Telecom Plattform“. Der Name deutet nur noch auf die Herkunft hin. Inzwischen ist OTP eine universell einsetzbare Standard-Library.

einer Erlang-Anwendung ein Supervisor-Tree aufgebaut. In einer solchen Struktur überwachen Supervisor Erlang-Prozesse, die selber wieder Supervisor sein können aber auch sogenannte Worker-Prozesse, die die eigentliche Arbeit erledigen. Supervisor bekommen von der Erlang-Runtime mitgeteilt, wenn sich ein Prozess unvorhergesehen beendet hat und starten ihn dann nach einem konfigurierbaren Vorgehen neu.

In einem WebServer wird in Erlang jeder Request von einem eigenem Erlang-Prozess abgewickelt. Stürzt in diesem Szenario ein Prozess ab, der sich um einen einzelnen Request kümmert, so ist es sinnvoll nur diesen einen Prozess neu zu starten. In diesem Fall ist sichergestellt, dass von dem Fehler auch nur ein einziger Request von dem Fehler betroffen ist. Alle anderen Requests werden vom System ohne Beeinflussung weiter bearbeitet. Kleine Fehler haben somit nur kleine Auswirkungen.

Tritt ein Fehler hingegen in einem zentralen Prozess innerhalb eines Anwendungsteils auf, so kann es unter Umständen sinnvoll sein, die von ihm abhängenden Prozesse neuzustarten.[4] Aber auch in diesem Fall laufen die anderen nicht betroffenen Anwendungsteile ohne Störungen weiter.

Als Nebeneffekt ergibt sich auch, dass resourcenhungrige Prozesse andere Prozesse nicht beeinflussen. Der Erlang-Scheduler sorgt dafür das jeder Prozess zu seinem Recht kommt. Auch die Garbage-Collection läuft isoliert nur für einen Prozess ohne die anderen Prozesse zu beeinflussen.

## Erlang-Prozesse

Erlang-Prozesse sind sehr leichtgewichtig. Durch das Erlang-Prozess-Modell wird jede Anwendung die Erlang-Prozesse verwendet automatisch nebenläufig. Da jeder Supervisor und jeder Worker in einem eigenen Erlang-Prozess laufen, sind OTP-Anwendungen direkt ohne weiteres hinzutun nebenläufig. Das ist der Grund wieso Erlang-Anwendung ausgezeichnet skalieren.

Durch das Erlang-Prozess-Modell ist es auch nicht mehr notwendig zu wissen, wo ein Erlang-Prozess läuft. Die Kommunikation zwischen lokalen und Erlang-Prozessen auf anderen Nodes ist vollkommen transparent für den Entwickler. Ganz egal ob diese Prozesse in der gleichen Erlang-VM (einem Erlang-Node), in einem anderen Erlang-Node auf der gleichen Maschine oder gar einem Erlang-Node auf einem anderen Host laufen, die Kommunikation mit ihnen verläuft immer nach dem selben Weg. Erlang unterstüzt damit bereits durch Sprach-Features die Kommunikation von Prozessen über Node- und host-Grenzen hinweg. Der Entwickler kann sich somit auf die Lösung des fachlichen Problems konzentrieren und muß keine Gedanken darüber machen, wie die Kommunikation zwischen Nodes zuverlässig funktionieren kann.

Auf einem Erlang-Node laufen in der Praxis hunderte, tausende oder mehrere Millionen Erlang-Prozesse.[5]

## Verteilte Anwendungen

Erlang geht aber noch einen Schritt weiter. „OTP-Applications“ erlauben eine Anwendungen auf mehreren Knoten zu starten. Ein Knoten wird hierbei als Hauptknoten definiert und auf diesem läuft zunächst die Anwendung. Kommt es nun zu einer Störung und die Anwendung auf diesem Knoten fällt aus, so bemerken das die Anwendungen im „standby“ auf den anderen Knoten und ein Ersatz-Knoten übernimmt automatisch ohne weiteres Zutun die Arbeit. Die Downtimes sind minimal. Ist die Anwendung auf dem Hauptknoten wieder verfügbar, so übernimmt der Hauptknoten wieder die Arbeit und die anderen Knoten gehen wieder in Standby.

## Weitere Sprach-Features

Erlang ist eine funktionale Programmiersprache. Funktionen lassen sich als Parameter an Funktionen übergeben.

Erlang besitzt eine ausgeprägte von Prolog beeinflußte Pattern-Matching-Syntax. Das in Erlang verwendete Pattern-Matching erlaubt mit wenigen Zeilen Fakten zu definieren. Durch die Definition von Fakten werden Programme kompakter.

Erlang Anwendungen werden – wie Anwendungen in anderen Sprachen auch – so erstellt, dass sie vom korrekten Verhalten der Umwelt ausgehen. Kommt es in einer Erlang-Anwendung jedoch zu einem nicht akzeptablen Fehlerfall, so stellt der einzelne Erlang-Prozess seine Arbeit ein. Der Supervisor-Tree (siehe

[4] Ob das wirklich sinnvoll und notwendig ist, muß im Einzelfall entschieden werden und der Supervisor-Tree entsprechend aufgebaut werden.

[5] Das ist auch nicht weiter verwunderlich, da Erlang-Prozess die Funktion von Objekten in objektorientierten Sprachen übernehmen. Zählen Sie einfach mal die Anzahl der Objekte in Ihrer aktuellen Anwendung.

oben) wird dann entsprechend seiner Vorgaben Prozesse neu starten. Einen try-catch-Block zu schreiben macht in diesem Zusammenhang auch überhaupt keinen Sinn. In diesem Zusammenhang hört man oft die Aussage „let it crash“ sei ein Design-Prinzip von Erlang. Das Einstellen der Arbeit ist jedoch das sinnvollste, was man in einer nicht hervorgesehenen Situation machen kann.

Erlang erlaubt den Austausch von Code während der Laufzeit. Damit wird es möglich eine Anwendung zu aktualisieren ohne jegliche Downtime.[6]

Anwendungsmonitoring: Die Erlang-Umgebung kennt zu jedem Prozess den letzten Funktionsaufruf und seine Parameter. Diese Parameter lassen sich abfragen und so zum Debugging verwenden. Mit dem mit Erlang mitgelieferten Observer läßt sich komfortabel der aktuelle Gesamtzustand des System analysieren.

## Resumé

Durch die aufgezählten Eigenschaft von Erlang und der Erlang-Umbebung eignet sich Erlang hervorragend für die Implementierung von verteilten hochverfügbaren Serversystemen mit hohen Zugriffszahlen. Server mit direkten Kundenkontakt sind ein Beispiel für ein solches Szenario. Kein Kunde wird für einfache Anfragen lange Wartezeiten oder gar einen Ausfall über einen längeren Zeitraum akzeptieren.

## Fun Facts

**Adam Kay**, der Schöpfer von SmallTalk und der Erfinder des Terminus „Objekt orientierte Programmierung“ (OOP) hat später gesagt, dass es ein Fehler war das Objekt durch diesen Terminus so in den Mittelpunkt zu stellen.. Es ginge vielmehr um den Austausch von Nachrichten zwischen den Objekten, die ihren Speicherbereich isolieren.

Das **WhatsApp**-BackEnd ist komplett in Erlang geschrieben und wird von einer Gruppe von ca. 20 Entwicklern weiterentwickelt und betrieben. Die Anwendung erstreckt sich über mehrere Erlang-Nodes, bei denen ca. 1 Millionen User gleichzeit in einem Node bedient werden. Nach Messungen von WhatsApp skaliert Erlang hierbei linear mit den Connections. [PDF].

[6] Auch wenn es meist einfacher ist, einen zweiten Node die Arbeit übernehmen zu lassen, so kann dieses Feature in Einzelfällen außerordentlich nützlich sein.

Das in Erlang programmierte Telefonvermittlungssystem AXD301 von Ericsson erreichte eine Verfügbarkeit von **99,9999999%** in einer Laufzeit von 20 Jahren. [Link]

Nach eigenen Aussagen liefert Cisco jährlich 2 Millionen Geräte mit Erlang-Code aus. Schätzungsweise **90% des Internettraffics** läuft über Router die in **Erlang** programmiert sind.

## Anwendungen in Erlang

- Amazon SimpleDB
- CouchDB / Couchbase
- RabbitMQ
- Riak

## Links

- Erlang Homepage (http://erlang.org/)
- t3n: „Die Programmiersprache, die Whatsapp antreibt“ (https://t3n.de/magazin/erlang-247504/)
- Erlang Companies (https://erlang-companies.org/)
- „Learn you some Erlang“ (https://learnyousomeerlang.com/)