Lars Fritsche
M-Nr: 1691285

TK1 – Theorieübung 1

Aufgabe 1.

1. Problem: In einem verteilten System ist der globale Zustand nicht abrufbar, da man zwar in der Lage ist einzelne Bestandteile des Systems nach ihrem Zustand zu befragen, jedoch ist diese Information nicht zu 100% zuverlässig, da die Information bei Empfang bereits veraltet ist und somit nicht zwingend den aktuellen Zustand wiederspiegelt.
2. Problem: Die Uhren sind nicht synchron, da zwar Protokolle wie NTP existieren, die die Zeit verschiedener Teilnehmer eines Netzes synchronisieren, jedoch ist dies immer mit einer gewissen Ungenauigkeit verbunden, die nicht überwunden werden kann. Events die auf verschiedenen Teilen des Systems stattfanden, lassen sich so nicht mit absoluter Sicherheit vergleichen, da es möglich ist, dass für zwei Events E1 und E2 (von verschiedenen Systemen) $t(E1) > t(E2)$ ausgewertet wird, obwohl $t(E1) < t(E2)$ gilt.
3. Problem: Die Ausführung ist nicht deterministisch, da man zum Beispiel nie mit absoluter Gewissheit sagen kann, wann welche Nachricht und in welcher Reihenfolge empfangen wird, wenn zwei Sender an den gleichen Empfänger senden. Gleiches gilt für die Ausführung mit mehreren Threads oder wenn verschiedene Teile des verteilten Systems ausfallen.

Aufgabe 2.

1. Die erste Abstraktionsebene ist die physikalische Konfiguration des verteilen Systems, bei dem es um die genauen Details der einzelnen Systeme, wie Hardware, sowie die Einzelheiten der Verbindungen geht.
2. Die zweite Abstraktionsebene ist die logische Konfiguration, bei der physikalische Details, wie die verwendete Hardware, wegabstrahiert werden. Im Vordergrund stehen die eigentlichen Verbindungen der einzelnen autonomen Systeme, bei denen es nicht mehr entscheidend ist, wie diese Verbindung realisiert wurden. Diese Ebene veranschaulicht das „communication subsystem“, das die einzelnen autonomen Systeme miteinander vernetzt.
3. Die dritte Abstraktionsebene ist das Prozessnetzwerk, bei dem die eigentliche Topologie nicht mehr entscheidend ist. Knoten in diesem Modell stellen Prozesse dar, die Nachrichten mit anderen Prozessen austauschen. Dabei werden keine Aussagen über Zeitbedinungen getroffen, da das Modell nur den Nachrichtenverlauf modelliert.
4. Die vierte Abstraktionsebene modelliert den verteilten Algorithmus, für den es unerheblich ist, was das zugrunde liegende System, die benutze Programmiersprache oder die Topologie des Netzes ist. Das Modell beschreibt Zeitbedinungen, die einzelne Prozesse einhalten müssen.

Aufgabe 3.

„Transparency“ bedeutet im Kontext verteiler Systeme, das Verstecken oder Verhüllen von bestimmten Details des Netzes, wie zum Beispiel das Verhalten bei Ausfällen oder Skalierbarkeit und den Zugriff auf entfernte Ressourcen. Die Verantwortung für die Aufrechterhaltung der Systemstruktur, sowie der Zugriff auf einzelne Teile, soll von einem Framework getragen werden, dessen Aufgabe es ist den Service, der auf ihm definiert wird, aufrechtzuerhalten.

Beispiele:

Performance und Scaling transparency:

Google App Engine – je nach Last steigt oder sinkt die Performance, darüber hinaus ist es möglich sehr viele Instanzen zu starten, ohne dass ein Verlust der Performance bemerkbar ist.

Mobility transparency:

Das Wechseln mit seinem Handy von einer Funkzelle in eine andere, ohne dass der Anruf abricht, da er weitergereicht wird.

Failure transparency:

Skype nutzt ein P2P Netzwerk um seinen Service bereitzustellen, wobei einzelne Teilnehmer ausfallen können, ohne das Netz zu gefährden.

Location transparency:

Für den Nutzer ist es nicht ersichtlich, wo die Daten in der Dropbox hinterlegt sind bzw. auf wievielen Servern sie verteilt sind

Aufgabe 4.

1. Distributed operation system approach – die Unterstützung für die Programmierung von verteilten Systemen soll auf Betriebssystemebene erfolgen. Der Nachteil ist hierbei, das große Systeme für gewöhnlich heterogen sind und eine derartige Lösung eine uniforme Lösung erzwingt, die bei einer höheren Skalierung zum Problem werden kann.
2. Distributed database approach – anstelle der Unterstützung auf Betriebssystemebene, wird bei diesem Ansatz ein Datenbanksystem eingesetzt. Der Vorteil dabei ist, dass alle gängigen Datenbankeigenschaften unterstützt werden, jedoch treten auch einige Probleme mit einer derartigen Lösung auf. Zum Beispiel lassen sich bestimmte Algorithmen für verteilte Systeme nicht mit einer solche Realisierung verwirklichen.
3. Protocol approach for dedicated purposes – für die Verbindung werden standardisierte Protokolle genutzt, die zwar Heterogenität in einem Netz erlauben, jedoch auf Standardfunktionalitäten beschränkt sind.
4. Distributed Programming Language approach – zuerst wurde dieser Ansatz durch spezialisierte Programmsprachen für verteilte Systeme realisiert, für die spezielle Compiler den Programmierer unterstützten. Der heutige Ansatz nutzt weiter verbreitete Programmiersprachen zusammen mit einer Middleware, die als „distributed runtime system“ agiert und als Schnittstelle zwischen den verteilten Prozessen und dem eigentlichen verteilten System dient. Der Vorteil dieses Konzepts ist die hohe Heterogenität, die erzielt werden kann, sowie die Nutzung von verschiedenen Programmiersprachen. Dies kann jedoch ebenso zum Nachteil werden, da die Nutzung von mehreren Programmiersprachen ebensoviele Distributed Run Time Systems benötigt. Die Nutzung von vielen verschiedenen Programmiersprachen führt außerdem dazu, dass Programmierer einen hohen Lernaufwand aufwenden müssen, um solche Systeme zu verstehen. Weiterhin kann die Unterstützung von vielen verschiedenen Systemen dazu führen, dass die Effizienz unter diesem Aspekt leidet, da Lösungen nicht mehr speziell auf bestimmte Systeme zugeschnitten werden und allgemeine Lösungen auf verschiedenen Systemen zu unterschiedlichem Performanceverhalten führen kann.