

Lastverteilung

Verteilte Softwaresysteme

Prof. Dr. Oliver Braun

Letzte Änderung: 10.12.2018 21:11

- Regelung der konkurrierenden Zugriffe auf die Ressourcen
- zu verteilende Objekte sind die Prozesse der Anwendungen
 - der Begriff Tasks wird in dem Zusammenhang synonym genutzt
- Unterscheidung von Prozessen:
 - **lokaler Prozess**: wird dort ausgeführt, wo er gestartet wurde
 - **Remote-Prozess**: können durch Lastverteilung ausgelagert werden
- **reguläre Prozesse** können nur lokal ausgeführt werden
- **generische Prozesse** können verteilt ausgeführt werden
- Synonyme für die Rechenressourcen:
 - **Prozessor, Rechner, Knoten**

- Minimierung der Kosten der Interprozesskommunikation
- Erreichung eines hohen Speedups
- Effiziente Ausnutzung aller Prozessoren
- minimale Bearbeitungszeiten für Prozesse
- Minimierung der totalen Ausführungskosten

- keine Verwendung von Vorwissen über Prozesse
- dynamisch, basierend auf Systemzustand
- schnelle Algorithmen
- geringer Overhead für Beschaffung von Systeminformationen über Zustand
- Stabilität des Systems, Aufwand für Lastausgleich gering ggü Rechenprozesse
- Skalierbar bei Hinzunahme neuer Knoten
- Ortstransparent
- hohe Ausfallsicherheit
- Fairness ggü Nutzern, d.h. Besitzer eines Rechners hat Vorrang

Statische Lastverteilung

- vor dem Ausführen verteilen
- nutzen auch **Anforderungsprofil** der Prozesse, z.B. an Prozessoren, Speicher, etc.
- besonders wichtig ist die Kenntnis von Ausführungszeiten, sowie der Speedup-Charakteristik
- verschiedene Stufen von Vorwissen
 - kein Vorwissen
 - Verteilung der Ausführungszeiten im gesamten Workload bekannt, aber keine Informationen über einzelne Jobs
 - Jobs sind Klassen mit spezifischen Eigenschaften zugeordnet
 - Ausführungszeiten für Jobs für jede beliebige Rechnerkombination liegen vor

- statische Lastverteilung ist ein Scheduling-Problem
- Ablaufplan muss erstellt werden
- **Queuing-Systeme**
 - verteilen Prozesse auf verschiedene Queues, ohne Startzeiten festzulegen
- **Plannende Systeme**
 - vollständige Ablaufpläne mit Startzeiten und Zuweisung von Zeitintervallen auf Rechnern an Jobs
- Ausführbarkeit eines Schedules (*feasibility*) ist gegeben, wenn es
 - keine Überlappungen von Zeitintervallen für denselben Job
 - oder auf demselben Rechner gibt
- **at-most-once-schedule**: keine Umplanung möglich
- **multiple-schedule**: Umplanung möglich

- für die Zuweisung von m Prozesse auf q Prozessoren gibt es q^m Möglichkeiten
 - wenn keine sonstigen Einschränkungen vorliegen
- optimale Zuordnung kann, außer in einfachen Fällen, nicht in polynomialer Zeit gefunden werden
 - NP-schweres Problem

Statische Lastverteilung: Jobmodelle

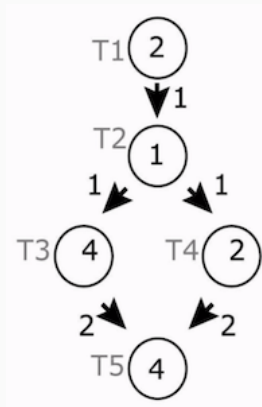
1. Task-Präzedenz-Graphen
2. Task-Interaktionsgraphen
3. Workflows
 - wesentlicher Teil ist die **Partitionierung** der Graphen
 - Zerlegung in Gruppen von Tasks, die einem Prozessor zugeordnet werden
 - gegenläufige Ziele
 - Minimierung des Overheads der Interprozesskommunikation
 - Parallelisierungspotential soll so gut wie möglich ausgeschöpft werden

- sei $P = \{P_1, P_2, \dots, P_n\}$ die Menge von Prozessen, die auf einer Menge von identischen Prozessoren ausgeführt werden sollen
- gegeben sei eine partielle Ordnungsrelation $<$ auf P
 - entspricht der zeitlichen Reihenfolge oder Präzedenzordnung
- $(P, <)$ wird beschrieben durch den Graph $G = (V, A)$ mit
 - Knoten V repräsentieren die Prozesse
 - gerichtete Kanten A stellen die Präzedenz zwischen den Knoten dar

- den Knoten $u \in V$ wird als Kostenfunktion $W(u)$ die Ausführungszeit der Tasks zugeordnet
- den gerichteten Kanten $(u, v) \in A$ ist eine Kostenfunktion $w(u, v) = (l, l')$ zugeordnet mit
 - l = Kommunikationskosten, falls u und v verschiedenen Prozessoren zugeordnet sind
 - l' = Kommunikationskosten, falls u und v dem gleichen Prozessor zugeordnet sind
- sind die Kosten l' vernachlässigbar klein gegenüber den Kosten l , nimmt man üblicherweise nur $w(u, v) = l$

Beispiel

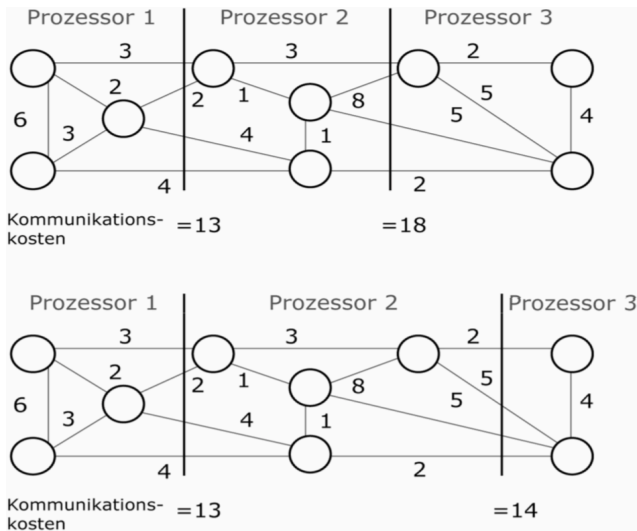
- 5 Tasks mit Ausführungszeiten, Abhängigkeiten und Kommunikationskosten
- Wie könnte der Ablaufplan aussehen?



Quelle: Bengel et. al, Masterkurs Parallele und Verteilte Systeme

- zeitlicher Ablauf und zeitliche Abhängigkeiten bleiben unberücksichtigt
- die Kanten stellen Interaktion zwischen zwei kommunizierenden Prozessen dar
- jede Kante $(u, v) \in A$ hat Kantengewicht $w(u, v)$, die Kommunikationskosten darstellen, wenn u und v auf verschiedenen Prozessoren ausgeführt werden
- Knotengewicht $w(u)$ repräsentiert die Ausführungszeit als Kostenfunktion
- verschiedene Algorithmen
- ein einfacher:
 - man finde minimale Schnitte um die Kommunikationskosten zu minimieren, ohne die Prozessoren zu überlasten

Beispiel



Quelle: Bengel et. al, Masterkurs Parallele und Verteilte Systeme

- Erweiterung der Jobmodelle
- insbesondere für den Einsatz im Grid Computing
- oft als DAGs oder als Petrinetze
- komplexere Workflows enthalten Schleifen und bedingte Anweisungen und können dann **dynamisch** stückweise in DAGs überführt werden
- Beispiele
 - DAGMan für Lastverwaltungssystem Condor
 - BPEL bzw. BPEL4WS
 - Abstract Grid Workflow Language (AGWL)
 - Grid Workflow Definition Language (GworkflowDL)

Statische Lastverteilung: Lösungsverfahren

- abhängig von den Jobs und von deren Anforderungen
- abhängig von Rechnerumgebung, z.B. ob homogen oder heterogen
- es existiert eine Vielzahl von Optimalitätskriterien (Zielfunktionen)
- im allgemeinen kein optimales Lösungsverfahren

aus Anwendersicht:

- Gesamtbearbeitungszeit (Makespan)
 - = Fertigstellungszeit des letzten Tasks
- Anzahl der Überschreitungen einer Deadline
- Summe der gewichteten Fertigstellungszeiten
- Summe der gewichteten Antwortzeiten
 - = Fertigstellungszeiten, gerechnet ab einer frühestmöglichen Startzeit
- Summe der gewichteten Verzugszeiten

aus Ressourcensicht:

- Auslastung / Grad der Parallelisierung

- dynamische Programmierung
- lineare Programmierung
- Branch-and-Bound-Algorithmus
- lokale Suchverfahren
- Simulated Annealing
- Tabu Search
- Evolutionäre Algorithmen
- ...

Dynamische Lastverteilung

- zwei Beispiele zur Verdeutlichung:
- Workstation-Modell
 - Workstations mehrere Benutzer sind vernetzt
 - Workstation ist entweder benutzt oder unbenutzt und könnte Berechnungen übernehmen
 - Lastverteilung so, dass Prozesse auf unbenutzte Workstations ausgelagert werde
 - Rückkehrende Benutzer haben aber Vorrang auf eigener Workstation
- Prozessorpool
 - zentraler Server nimmt Prozesse von Benutzern entgegen und verteilt sie auf Rechner im Pool
 - Benutzer erhalten eine passende Anzahl von Prozessoren, je nach Parallelisierungsgrad der Anwendung

- die Aufgaben bei der dynamischen Lastverteilung entsprechen den Aufgaben in einem **Regelkreis**
- **Last erfassung** oder **Lastmessung** (Messfühler)
- **Lastbewertungskomponente** (Regler)
- **Lastverschiebekomponente** (Stellglied)

- **anwendungsintegrierte** vs. **systemintegrierte** Lastverwaltung
- **Load Sharing** vs. **Load Balancing**
 - Load Sharing – Ziel: kein Rechner ungenutzt, solange passende Prozesse warten
 - Load Balancing – Ziel: darüber hinaus noch möglichst ausgeglichen verteilt
- bei Load Balancing mehr Tasktransfers \Rightarrow Kosten/Nutzen abschätzen
- Lastverwaltung
 - minimal: nur bzgl. direkter Nachbarn
 - maximal: gesamtes System

- **Initialisierungsstrategie** (*Initiation Policy*)
 - legt fest von welchem Knoten die Initiative zur Lastverteilung ausgeht
- **Informationsstrategie** (*Information Policy*)
 - bestimmt wann Information ausgetauscht wird
 - von welchem Knoten angefordert bzw. verbreitet wird
 - welche Informationen wichtig sind
- **Transferstrategie** (*Transfer Policy*)
 - entscheidet ob Lastausgleich erforderlich ist
- **Selektionsstrategie** (*Selection Policy*)
 - wählt zu transferierenden Prozess aus
- **Lookationsstrategie** (*Location Policy*)
 - bestimmt, welcher Knoten an einem als erforderlich erkannten Lastausgleich beteiligt werden soll

- zentrale Komponente mit den Strategien
 - Initialisierungsstrategie ist nicht notwendig
- Lastmessung erfolgt auf den einzelnen Knoten
- einfaches Beispiel: Prozessorfarm zum parallelen Rechnen
 - ein Master führt die sequentiellen Teile durch
 - die Worker fordern beim Master Arbeit an, dieser verteilt
- zentraler Knoten ist Flaschenhals
- dadurch skaliert der Ansatz schlecht
- ist aber einfacher zu implementieren

- mehrere oder sogar alle Knoten verfügen über Lastbewertung
- gemischte Verfahren halten z.B. Informationen zentral, während Bewertung und Lastausgleich dezentral erfolgt
- kooperative und nicht-kooperative Verfahren
- kooperative sind komplexer, da die Lastverteilungseinheiten miteinander kommunizieren um einen gemeinsamen Algorithmus auszuführen

- eher Load Sharing statt Load Balancing
- Prozessor der Prozesse bekommt, verteilt diese weiter
- sehr viel einfacher zu implementieren, da Prozesse nie im Laufenden Zustand migriert werden
- Beispiel: Zufallsalgorithmus
 - jeder Prozessor schickt die Tasks an zufällig ausgewählte Knoten weiter
 - dadurch ist die Wahrscheinlichkeit einen Prozess zu erhalten für jeden Knoten gleich

- neue Tasks optimal platzieren und
- Last von einem überlasteten Knoten weg zu transferieren
- Migration in dezentralen Systemen ermöglicht den höchsten Grad an Lastausgleich

Vorteile von Systemen mit der Möglichkeit zur Migration

- verringern von Laufzeiten durch parallele Prozessausführung oder Transferieren auf schneller CPUs
 - Achtung: Kosten/Nutzen
- erhöhen des Durchsatzes durch Nutzung freier Kapazitäten
- Reduktion der Netzwerkbelastung durch kürzere Wege
- Verlässlichkeit erhöhen
 - kritische Prozesse auf verlässliche Rechner verlegen oder Kopien anlegen und transferieren
- Bewältigung von Lastspitzen oder heterogener Lastverteilung
- Transferieren bei Shutdown oder wenn Besitzer der Rechner eigene Jobs startet
- Transfer auf Rechner mit höheren Sicherheitsanforderungen
- vermeiden von Verhungern (starvation) von Prozessen

- Transparenz bzgl. Objektnamen und -orten, Systemaufrufen und IPC
- Transparenz gegenüber dem Nutzer
- möglichst geringe Kosten für Transfer
- keine Restabhängigkeit zu altem Knoten
 - Zugriffe mit höherer Last verbunden
 - Fehlertoleranz nimmt ab

- **Einfrieren** des Prozesses
 - Prozess muss entweder sofort oder nach einem nicht-unterbrechbaren Systemaufruf blockiert werden
 - schnelle I/O-Operationen werden zu Ende geführt
 - bei langsamen muss korrekte Weiterführung nach dem Neustart sichergestellt werden
 - Informationen über geöffnete Dateien müssen festgehalten werden

- **Transfer des Adressraums** und der Zustandsinformationen

3 mögliche Methoden

- **vollständiges Einfrieren**
 - Prozess bleibt während der gesamten Prozedur gestoppt
 - es kann zu Timeouts kommen und Benutzer kann beeinträchtigt werden
- **Vortransferierung** (*pre-copying*)
 - transferiert den Adressraum des noch laufenden Prozesses
 - Seiten, die der Prozess noch modifiziert, werden sukzessive nachgeladen bis nur noch wenige übrig sind
- **Transfer bei Referenzierung** (*copy-on-reference*)
 - Adressraum wird auf dem Quellknoten zurück gelassen und der Prozess auf dem Zielknoten gestartet
 - bei Bedarf werden die Seiten sukzessive nachgeladen
 - durch die weitere Abhängigkeit vermindert sich aber die Fehlertoleranz

- **Nachrichtenweiterleitung**

3 Arten von Nachrichten

- Nachrichten, die während der Prozess eingefroren ist, am Quellknoten eintreffen
- Nachrichten, die nach Weiterführung auf dem Zielknoten, am Quellknoten eintreffen
- Nachrichten, die nach Weiterführung erst versendet werden

4 Methoden um mit den Nachrichten zu verfahren

- **Rücksendung**, mit Maßgabe erneut zu senden und dazu den Empfänger zu finden
- **Weiterleitung vom Ursprungsknoten**, Ursprungsknoten muss über weitere Migration informiert werden
- **Wiederholte Weiterleitung**, durch alle Knoten auf denen der Prozess je war
- **Aktualisierung** von potentiellen Sendern

- Ermöglichen der Fortführung der Kommunikation mit Koprozessen

zwei grundsätzliche Möglichkeiten

- **Verbot der Trennung**
 - Eltern- und Kindprozesse müssen gemeinsam migriert werden
 - dadurch kann Parallelisierungspotential nicht genutzt werden
- **Kommunikation über den Ursprungsknoten**
 - bekannte Nachteile: erhöhte Last, verringerte Fehlertoleranz
- **Aktivieren des Prozesses auf dem Zielknoten**
- das Verfahren funktioniert in homogenen Umgebungen
- in heterogenen Umgebungen sehr viel komplexer und sprengt hier den Rahmen