Verteilte Sensornetzwerke

Mit Datenaggregation und Sensorfusion

PD Stefan Bosse

Universität Bremen - FB Mathematik und Informatik

Verteilte Programmierung von Sensornetzwerken

Neben P2P Kommunikation, RPC, und Gruppenkommunikation gibt es höhere Netzwerkorganisationen und Softwareframeworks

Übergang zu heterogener Cluster- und Cloud pogrammierung

Grundlagen von Map&Reduce Verfahren

Einführung in das MPI Softwareframework

Parallele und Verteilte Architekturen

Functionality, Applications	Computer Clusters [10,28,38]	Peer-to-Peer Networks [34,46]	Data/ Computational Grids [6,18,51]	Cloud Platforms [1,9,11,12,30]
Architecture, Network Connectivity, and Size	Network of compute nodes interconnected by SAN, LAN, or WAN hierarchically	Flexible network of client machines logically connected by an overlay network	Heterogeneous clusters interconnected by high-speed network links over selected resource sites	Virtualized cluster of servers over data centers via SLA
Control and Resources Management	Homogeneous nodes with distributed control, running UNIX or Linux	Autonomous client nodes, free in and out, with self-organization	Centralized control, server-oriented with authenticated security	Dynamic resource provisioning of servers, storage, and networks
Applications and Network-centric Services	High-performance computing, search engines, and web services, etc.	Most appealing to business file sharing, content delivery, and social networking	Distributed supercomputing, global problem solving, and data center services	Upgraded web search, utility computing, and outsourced computing services
Representative Operational Systems	Google search engine, SunBlade, IBM Road Runner, Cray XT4, etc.	Gnutella, eMule, BitTorrent, Napster, KaZaA, Skype, JXTA	TeraGrid, GriPhyN, UK EGEE, D-Grid, ChinaGrid, etc.	Google App Engine, IBM Bluecloud, AWS, and Microsoft Azure

Abb. 1. Vergleich verschiedener paralleler und verteilter Berechnungssysteme [10]

Cluster Computing

- Ein Computer-Cluster besteht aus miteinander verbundenen eigenständigen Computern, die kooperativ als eine einzige integrierte Computer-Ressource arbeiten.
- In der Vergangenheit haben Computer-Cluster eindrucksvolle Ergebnisse bei der Bewältigung hoher Arbeitslasten mit großen Datenmengen gezeigt.

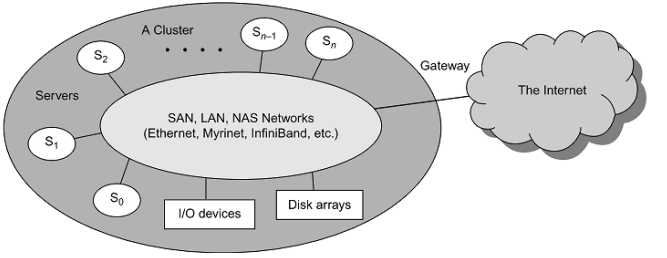


Abb. 2. Cluster von Servern mit Disk Arrays, Hochgeschwindigkeitsverbindung, Ein- und Ausgabegeräte, und Ankopplung an das Internet [10]

Cluster Computing

- Ein idealer Cluster mit mehrere Systembilder sollte zu einem Single-System-Image (SSI) zusammengeführt werden.
- Cluster-Entwickler wünschen ein Cluster-Betriebssystem oder eine Middleware, um SSI auf verschiedenen Ebenen zu unterstützen, einschließlich der gemeinsamen Nutzung von CPUs, Arbeitsspeicher und E/A über alle Cluster-Knoten hinweg.
- Ein SSI ist eine durch Software oder Hardware erzeugte Illusion, die eine Sammlung von Ressourcen als eine integrierte Ressource darstellt.
- SSI lässt den Cluster wie eine einzige Maschine für den Benutzer erscheinen. Ein Cluster mit mehreren Systemabbildern ist nichts anderes als eine Sammlung unabhängiger Computer &arr; One Big Virtual Machine!

Cloud Computing

- Cloud Computing bedeutet u.A. die Bereitstellung von virtualisierten Ressourcen von Rechenzentren zu einer Internet-Cloud, die mit Hardware, Software, Speicher, Netzwerk und Services für bezahlte Benutzer zur Verfügung gestellt wird, um ihre Anwendungen auszuführen.

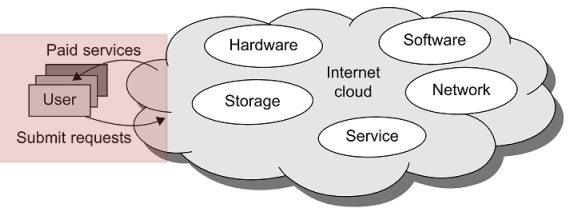


Abb. 3. Nutzer können virtualisierte Ressourcen anfragen → Auslagerung von Berechnung!

Virtuelle Netzwerke

- Neben der Virtualisierung von Ressourcen wie Speicher und CPUs ist die Virtualisierung und Transformation der Netzwerkstrukturen zentrale Methodik

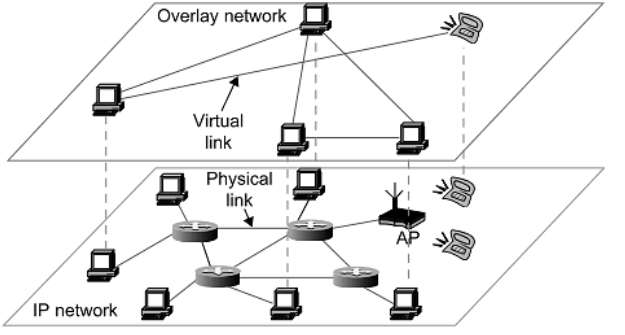


Abb. 4. Die Struktur eines P2P-Systems durch Abbildung eines physischen IP-Netzwerks auf ein Overlay-Netzwerk mit virtuellen Verbindungen. [10]

Partitionierung und Kommunikation

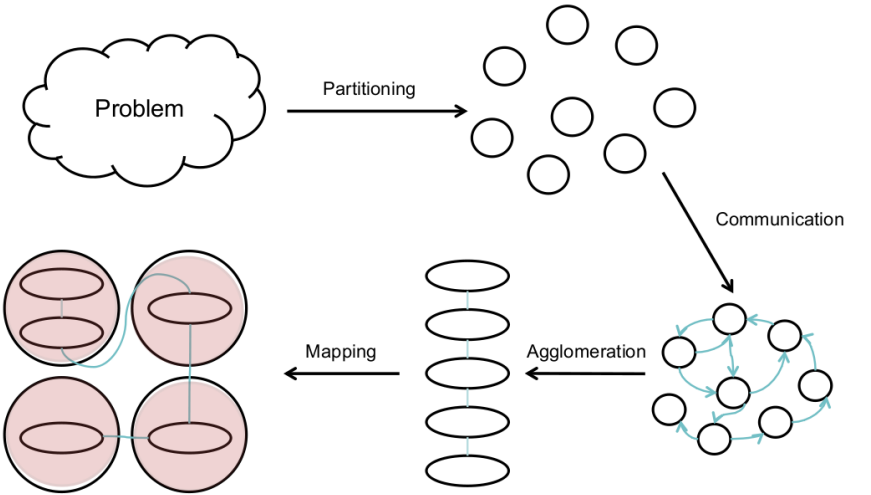


Abb. 5. Problemzerlegung durch Partitionierung → Abbildung auf Kommunikationsnetzwerke [Janetzko, MPI Practice 2014]

Gustafson Gesetz

- In parallelen Systemen drückt das Amdahlsche Gesetz den maximalen Speedup in Abhängigkeit vom sequenziellen Programmteil (bzw. dessen Ausführung) aus
  - Ein hoher Speedup und daher eine effiziente Ausnutzung (Workload) aller Verarbeitungseinheiten/Rechner ist nur durch Minimierung des sequenziellen Teils η (verursacht durch Kommunikation) möglich

Um bei Verwendung eines großen Clusters eine höhere Effizienz zu erzielen, muss die Größe des Problems so skalieren, dass sie der Cluster-Fähigkeit entspricht!

- Problem bei der Anwendung des Amdahlschen Gesetzes: Ein **konstanter Workload** wird für den sequenziellen und parallelen Anteil angenommen!
- Dies führt zu dem folgenden Beschleunigungsgesetz, das von John Gustafson (1988) vorgeschlagen wurde und als skalierte Arbeitslastbeschleunigung bezeichnet wird.

Gustafson Gesetz

- Sei *W* die Arbeitslast in einem bestimmten Programm. Bei Verwendung eines n-Prozessor-Systems skaliert der Benutzer die Arbeitslast zu *W**= η*W* + (1-η) *n****W*.
  - Nur der parallelisierbare Teil der Auslastung wird im zweiten Ausdruck n-mal skaliert.
- Diese skalierte Arbeitslast *W** ist im Wesentlichen die sequentielle Ausführungszeit auf einem einzelnen Prozessor.
- Die parallele Ausführungszeit einer skalierten Arbeitslast *W** auf *n* Prozessoren wird wie folgt durch eine skalierte Arbeitslast-Beschleunigung definiert:

$$S^\star = W^\star / W = \left[\eta W + (1-\eta) n W\right] / W = \eta + (1-\eta) n$$

- Amdahls und Gustafsons Gesetze werden bei unterschiedlichen Workloads *W* angewendet!

Map & Reduce

- Ein Web-Programmiermodell für die skalierbare Datenverarbeitung in großen Clustern über große Datenmengen.
- Das Modell wird häufig in Web-Scale-Search- und Cloud-Computing-Anwendungen eingesetzt.
- Aber auch funktionale und parallele Programmierung macht von MapReduce Methoden Gebrauch

Methode

- Es wird eine Map-Funktion angegeben, um eine Gruppe von Schlüssel/Wert-Zwischenpaaren zu generieren.
- Dann wird eine Reduce-Funktion auf diesen Paaren angewendet, um alle Zwischenwerte mit demselben Zwischenschlüssel zusammenzuführen.
- MapReduce ist hochgradig skalierbar, um hohe Parallelitätsgrade auf verschiedenen Arbeitsebenen zu erreichen.

Map & Reduce

- Ein typischer MapReduce-Berechnungsprozess kann Terabytes an Daten auf Zehntausenden oder mehr Client-Computern verarbeiten. Hunderte von MapReduce-Programmen können gleichzeitig ausgeführt werden.
  - Tatsächlich werden jeden Tag Tausende von MapReduce-Jobs in Clustern von Google ausgeführt.
  - Das Hadoop Framework bietet für WEB Anwendungen MapReduce Services an → Master-Slave Architektur!

Die Map-Funktion verarbeitet ein Paar (Schlüssel, Wert) und gibt eine Liste von Zwischenpaaren (Schlüssel, Wert) zurück:

$$map(k_1, v_1) \rightarrow list(k_2, v_2)$$

Die Reduzierungsfunktion führt alle Zwischenwerte zusammen, die die gleichen Zwischenschlüssel haben:

$$reduce(k_2, list(v_2)) \ \rightarrow \ list(v_3)$$

Map & Reduce

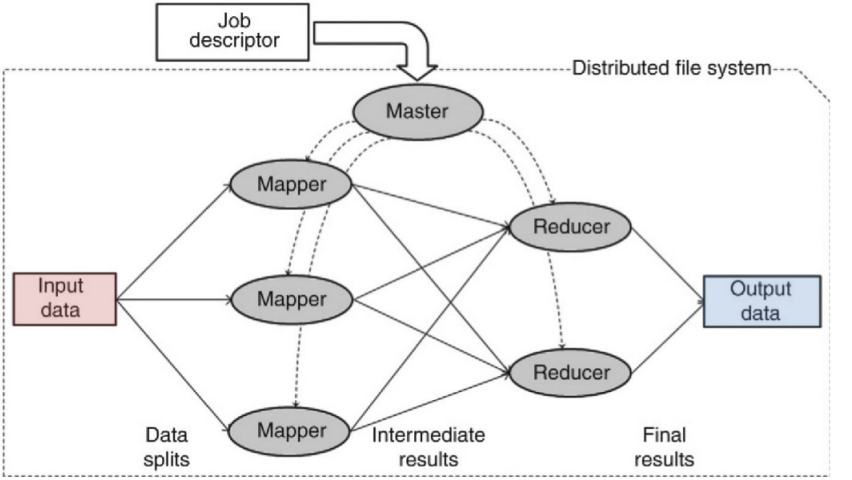


Abb. 6. Ausführungsphasen einer generischen MapReduce Applikation

Map & Reduce

MapReduce Phasen

1. Ein Master-Prozess erhält einen Jobdeskriptor, der den auszuführenden MapReduce-Job angibt. Der Jobdeskriptor enthält neben anderen Informationen den Ort der Eingabedaten, auf die unter Verwendung eines verteilten Dateisystems zugegriffen werden kann.

2. Gemäß dem Jobdeskriptor startet der Master eine Anzahl von Mapper- und Reducer-Prozessen auf verschiedenen Maschinen. Gleichzeitig startet es einen Prozess, der die Eingabedaten von seinem Speicherort liest, diese Daten in eine Gruppe von Aufteilungen unterteilt und diese Aufteilungen an verschiedene Zuordner verteilt.

3. Nach dem Empfang seiner Datenpartition führt jeder Zuordnungsvorgang die Zuordnungsfunktion aus (die als Teil des Jobdeskriptors bereitgestellt wird), um eine Liste von Zwischenschlüssel / Wert-Paaren zu erzeugen. Dann werden diese Paare auf der Basis ihrer Schlüssel gruppiert.

Map & Reduce

4. Alle Paare mit den gleichen Schlüsseln sind dem gleichen Reduziervorgang zugeordnet. Daher führt jeder Reduzierprozess die Reduktionsfunktion (definiert durch den Jobdeskriptor) aus, die alle Werte vereinigt, die mit dem gleichen Schlüssel assoziiert sind, um einen möglicherweise kleineren Satz von Werten zu erzeugen.

5. Dann werden die von jedem Reduktionsprozess erzeugten Ergebnisse gesammelt und an einen durch den Jobdeskriptor spezifizierten Ort geliefert, um die endgültigen Ausgabedaten zu bilden.

Map & Reduce

Beispiel in Lua

```
local options = {workers = 2}
local data = {34,35,36,37,38,39,40,41}

local function worker (id,set)
  local results = T{}
  for i = 1,#set do
    results:push(fib(set[i]))
  end
  return results
end

-- Parallel Processing
local Parallel = require('parallel')
local p = Parallel:new(data,options)
function sum (x,y) return x+y  end
p:time():
  map(worker):
  apply(function (r) print(r:print()) end):
  reduce(sum):
  apply(print):
  time()
```

MPI

Message Passing Interface: MPI

Ziele und Eigenschaften

- Anwendungsprogrammierschnittstelle (nicht unbedingt für Compiler oder eine Systemimplementierungsbibliothek).
- Effiziente Kommunikation:
  - Vermeidung von Arbeitsspeicher-Arbeitsspeicher Kopien
  - Überlappung von Berechnung und Kommunikation
  - Auslagerung auf Kommunikations-Coprozessoren, soweit verfügbar.
- Implementierungen, die in einer heterogenen Umgebung verwendet werden können (verschiedene Hostplattformen).
- Einfache Einbindung in Programmiersprachen und Bibliotheken und Plattformunabhängigkeit
- Zuverlässige Kommunikation: Nutzer/Programmier muss sich nicht um Kommunikationsfehler kümmern

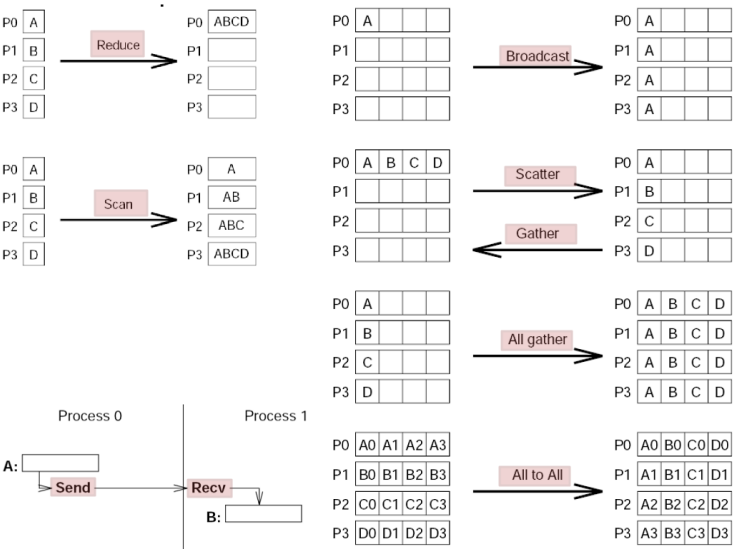
MPI

API

- Point-to-point communication,
- Datatypes,
- Collective operations,
- Process groups,
- Communication contexts,
- Process topologies,
- Environmental management and inquiry,
- The Info object,
- Process creation and management,
- One-sided communication,
- External interfaces,
- Parallel file I/O

MPI

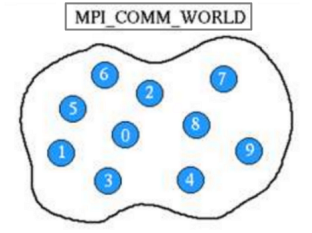
Operationen



MPI

Communicator

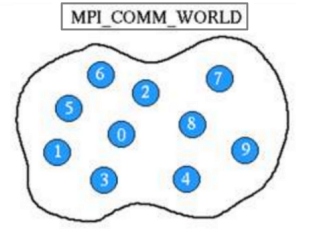
- Kommunikationswelt mit einer Gruppe aus Prozessen die gemeinsam Nachrichten austauschen können
- Nachrichten können nur innerhalb der Kommunikationswelt ausgetauscht werden
- MPI_COMM_WORLD ist die Standardwelt



MPI

Rank

- Einheitliche Prozessnummer innerhalb einer Kommunikationswelt
- Werden vom System bei der Initialisierung vergeben und werden fortlaufend ab 0 nummeriert
- Rank IDs werden bei der Kommunikation zur Identifikation von Empfänger und Sender verwendet (Kommunikationsendpunkte)
- Rank IDs dienen zur Programmdifferenzierung (*if rank==0 then do this else do that*)

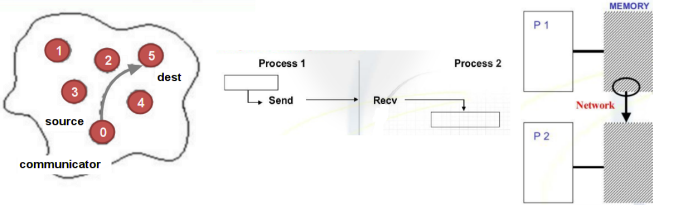


MPI

Point-to-Point Kommunikation

- Kommunikation zwischen zwei Prozessen
- Quellprozess sendet eine Nachricht (Typ, Daten) an einen Zielprozess unter Angabe der Rank ID
- Kommunikation kann nur innerhalb eines Communicators stattfinden

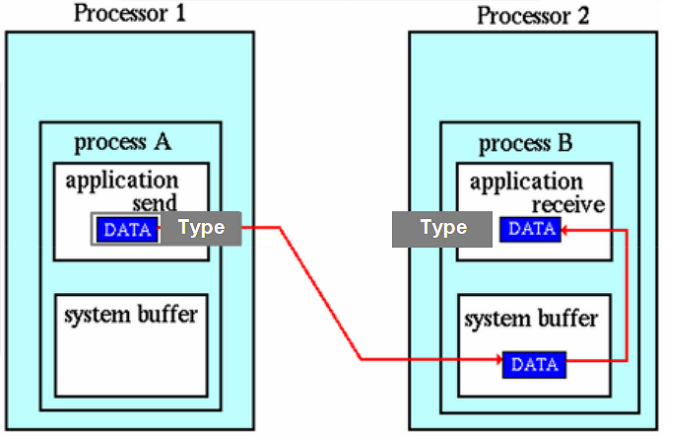
```
MPI.send(dest:number,message:{type:string,content:string})
```



MPI

- Damit der Zielprozess die Nachricht empfangen kann muss er einen Handler für den entsprechenden Nachrichtentyp einrichten:

```
MPI.recv(type:string,callback:function (message))
```



MPI

Broadcast Kommunikation

- Ein Quellprozess kann eine Nachricht an alle Prozesse innerhalb eines Communicators senden

```
MPI.broadcast(message:{type:string,content:string})
```