UNIVERSITY OF COPENHAGEN

Advanced Computer Systems (ACS)

DIKU Course Compendium

Block 2, 2015/16

Advanced Computer Systems (ACS)

DIKU Course Compendium

DIKU, Department of Computer Science,
University of Copenhagen, Denmark

Block 2, 2015/16

Contents

Preface

This compendium has been designed for the course Advanced Computer Systems (ACS), taught at the Department of Computer Science (DIKU), University of Copenhagen. The contents of the compendium are in correspondence with the rules of use at academic courses defined by CopyDan. The compendium is organized into 12 parts, each containing textbook chapters or reference papers related to topics covered in the course. Each part is also prefaced by a short description of the learning expectations with respect to the readings.

The compendium starts with a review of **fundamental abstractions** in computer systems, namely *interpreters*, *memory*, and *communication links* (Part 1). The course explores multiple **properties** that may be attached to these abstractions, and exposes principled design and implementation techniques to obtain these properties while respecting interfaces and achieving high performance. A first property is the notion of **strong modularity**, achieved by organization of interpreters into clients and services and use of remote procedure call (RPC) mechanics (Part 2).

After a brief review of general techniques for **performance** (Part 3), the properties of **atomicity** and **durability** are explored. The former property of atomicity may be understood with respect to before-or-after, or alternatively all-or-nothing semantics. Multiple different *concurrency control* protocols to achieve before-or-after atomicity over a memory abstraction are first introduced (Part 4). Following concurrency control, *recovery* protocols for all-or-nothing atomicity and durability are discussed (Part 5).

The text then ventures into a brief foray on techniques for **experimental design** (Part 6), which allow performance characteristics of different designs and implementations of a given abstraction to be analyzed. After this foray, the compendium then turns to the property of **high availability** in the presence of faults, achieved by a combination of techniques. First, general techniques for *reliability*, in particular replication techniques, are discussed (Part 7). Distribution of system functionality and replication introduce the problem of maintaining consistency. So, solutions for achieving high degrees of consistency in distributed scenarios, including *ordered multicast*, *two-phase commit*, and *state-machine replication*, are then discussed (Part 8). Finally, *communication* schemes that decouple system functions are discussed, along with the classic *end-to-end argument* (Part 9).

The text finally explores the property of **scalability** with large data volumes, and reviews design and implementation techniques for data processing operators, including *external sorting*, *basic relational operators and joins*, as well as *parallelism* (Parts 10, 11, and 12, respectively).

We hope you enjoy your readings!

Learning Goals

The learning goals for ACS are listed below.

Knowledge

- Describe the design of transactional and distributed systems, including techniques for modularity, performance, and fault tolerance.
- Explain how to employ strong modularity through a client-service abstraction as a paradigm to structure computer systems, while hiding complexity of implementation from clients.
- Explain techniques for large-scale data processing.

Skills

- Implement systems that include mechanisms for modularity, atomicity, and fault tolerance.
- Structure and conduct experiments to evaluate a system's performance.

Competences

- Discuss design alternatives for a modular computer system, identifying desired system properties as well as describing mechanisms for improving performance while arguing for their correctness.
- Analyze protocols for concurrency control and recovery, as well as for distribution and replication.
- Apply principles of large-scale data processing to analyze concrete information-processing problems.

Source List

G. Couloris, J. Dollimore, T. Kindberg. Distributed systems, concepts and design. Third Edition. Chapters 11 (except 11.2 and 11.3) and 13, pp. 419–423, 436–464, and 515–552 (72 of 772). Addison-Wesley, 2001. ISBN: 0201-61918-0

J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. Commun. ACM 53, 1, pp. 72-77 (6 of 159), 2010. Doi: 10.1145/1629175.1629198

D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. Commun. ACM 35, pp. 85-98 (14 of 1868), 1992. Doi: 10.1145/129888.129894

D. Lilja. Measuring Computer Performance: A Practitioner's Guide. Chapters 1, 2, and 6, pp. 1–24 and 82–107 (50 of 261). Cambridge University Press, 2000. ISBN: 978-0-521-64105-0

H. Garcia-Molina, J. D. Ullman, J. Widom. Database Systems: The Complete Book. Chapters 11.4 and 15, pp. 525–533 and 713–774 (71 of 1119). Prentice Hall, 2002. ISBN: 0-13-031995-3

G. Graefe. Encapsulation of parallelism in the Volcano query processing system. SIGMOD Rec. 19, 2, pp. 102-111 (10 of 632), 1990. Doi: 10.1145/93605.98720

D. Pritchett. BASE: An Acid Alternative. Queue 6, 3 (May 2008), pp. 48-55 (8 of 72), 2008. Doi=10.1145/1394127.1394128.

R. Ramakrishnan and J. Gehrke. Database Management Systems. Third Edition. Chapters 16–18, pp. 519–544, 549–575, and 579–600 (75 of 1065). McGraw-Hill, 2003. ISBN: 978-0-07-246563-1

J. H. Saltzer and M. F. Kaashoek. Principles of Computer System Design: An Introduction. Part I. Sections 2.1, 4.2, and 6.1, pp. 44–60, 167–172, and 300–316 (40 of 526). Morgan Kaufmann, 2009. ISBN: 978-0-12-374957-4

J. H. Saltzer and M. F. Kaashoek. Principles of Computer System Design: An Introduction. Part II. Chapters 8.1–4 and 8.6, pp 8-2 – 8-35 and 8-51 – 8-54 (38 of 826). Creative Commons License, 2009.

J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. ACM Trans. Comput. Syst. 2(4) pp. 277–288 (12 of 359), 1984. Doi: 10.1145/357401.357402

F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22(4) pp. 299–319 (21 of 409), 1990. Doi: 10.1145/98163.98167

A. S. Tanenbaum and M. V. Steen. Distributed Systems: Principles and Paradigms. Second Edition. Chapters 4, pp. 124–125 and 140–177 (40 of 686) Pearson International Edition, 2007. ISBN: 0-13-613553-6

Chapter 1

Fundamental Abstractions

This chapter contains the book chapter:

> J. H. Saltzer and M. F. Kaashoek. Principles of Computer System Design: An Introduction. Part I. Section 2.1, pp. 44–60 (17 of 526). Morgan Kaufmann, 2009. ISBN: 978-0-12-374957-4

The chapter reviews the **fundamental abstractions** in computer systems: memory, interpreters, and communication links. These abstractions manifest themselves in hardware and in software, in centralized as well as distributed systems. *The ultimate goal of this portion of the material is to convey the generality of these abstractions, and stimulate us to reflect on how different versions of these abstractions are implemented in terms of one another over different system layers.*

The learning goals for this portion of the material are listed below.

- Identify the fundamental abstractions in computer systems and their APIs, including memory, interpreters, communication links.
- Explain how names are used in the fundamental abstractions.
- Design a top-level abstraction, respecting its correspondent API, based on lower-level abstractions.
- Discuss performance and fault-tolerance aspects of such a design.

Chapter 2

Modularity through Clients and Services, RPC

This chapter contains the book chapter:

> J. Saltzer and M. F. Kaashoek. Principles of Computer System Design: An Introduction. Part I. Section 4.2, pp. 167–172 (6 of 526). Morgan Kaufmann, 2009. ISBN: 978-0-12-374957-4

The chapter discusses how to organize interpreters in terms of clients and services. An important property of this organization is **strong modularity**, the capability of bounding failure propagation between these components. A classic mechanism to achieve strong modularity with clients and services is the remote procedure call (RPC). *The ultimate goal of this portion of the material is to enable us to write strongly modular software with RPCs, and to reflect on how strong modularity affects program semantics but at the same time allows us to incorporate additional mechanisms (e.g., for scaling the number of connections) in-between modular clients and services.*

The learning goals for this portion of the material are listed below.

- Recognize and explain modular designs with clients and services.
- Predict the functioning of service calls under different RPC semantics and failure modes.
- Identify different mechanisms to achieve RPCs.
- Implement RPC services with an appropriate mechanism, such as web services.

Note that the course assignments will be instrumental in achieving the goals in this as well as other parts of the course.

This page has intentionally been left blank.

Chapter 3

Techniques for Performance

This chapter contains the book chapter:

> J. Saltzer and M. F. Kaashoek. Principles of Computer System Design: An Introduction. Part I. Section 6.1, pp. 300–316 (17 of 526). Morgan Kaufmann, 2009. ISBN: 978-0-12-374957-4

An important consequence of writing strongly modular software with clients and services is that service implementations may be optimized for performance or for reliability (or both) without affecting clients, as long as these optimizations do not affect semantics. **Performance** is a recurring theme in design and implementation of system abstractions and services. There are a number of general techniques for performance, among which *concurrency* is of special importance. *The ultimate goal of this portion of the material is to introduce us to metrics to characterize performance, and provide us with general techniques to design services for performance.*

The learning goals for this portion of the material are listed below.

- Explain performance metrics such as latency, throughput, overhead, utilization, capacity, and scalability.
- List common hardware parameters that affect performance.
- Apply performance improvement techniques, such as concurrency, batching, dallying, and fast-path coding.

Chapter 4

Concurrency Control

This chapter contains the book chapters:

> R. Ramakrishnan and J. Gehrke. Database Management Systems. Third Edition. Chapters 16–17, pp. 519–544, and 549–575 (53 of 1065). McGraw-Hill, 2003. ISBN: 978-0-07-246563-1

While concurrency is a powerful technique for performance, it is unfortunately hard to get concurrent software right. The properties of **atomicity** and **durability**, and in particular the atomicity variant of *before-or-after atomicity* (also called *isolation*), are crucial for correctness, but particularly challenging to achieve when concurrency is employed. Gladly, a general theory of how to write correct, concurrent software, which respects before-or-after atomicity, over a memory abstraction has been developed in the context of database systems. This chapter discusses this theory, which is grounded on the notions of transactions, the ACID properties, and the serializability of transaction schedules. The chapter elaborates on multiple protocols for concurrency control, based on locking, timestamps, and multi-versioning. *The ultimate goal of this portion of the material is to enable us to design our own strategies for achieving atomicity when writing performance-optimized, concurrent services, and reflect on their correctness by equivalence to a well-known concurrency control protocol.*

The learning goals for this portion of the material are listed below.

- Identify the multiple interpretations of the property of atomicity.
- Implement methods to ensure before-or-after atomicity, and argue for their correctness.
- Explain the variants of the two-phase locking (2PL) protocol, in particular the widely-used Strict 2PL.
- Discuss definitions of serializability and their implications, in particular conflict-serializability and view-serializability.
- Apply the conflict-serializability test using a precedence graph to transaction schedules.
- Explain deadlock prevention and detection techniques.

- Apply deadlock detection using a waits-for graph to transaction schedules.
- Explain situations where predicate locking is required.
- Explain the optimistic concurrency control and multi-version concurrency control models.
- Predict validation decisions under optimistic concurrency control.

Chapter 5

Recovery

This chapter contains the book chapter:

> R. Ramakrishnan and J. Gehrke. Database Management Systems. Third Edition. Chapter 18, pp. 579–600 (22 of 1065). McGraw-Hill, 2003. ISBN: 978-0-07-246563-1

Atomicity and **durability** are challenging in important ways other than with concurrency. First, we must deal with the risk of partial execution of functions in our service, e.g., due to errors, leading to data corruption and compromising *all-or-nothing atomicity.* Second, if volatile memory is employed, even complete execution of a function may not guarantee durability of its effects. Third, performance optimizations, such as use of two-level memories, may complicate the determination of the current state of the system in the event of a failure. Fortunately, methodologies to achieve atomicity and durability in the face of these challenges have been developed in the context of database systems. Log-based recovery techniques allow us to survive both fail-stop and media failures. These techniques have been heavily studied, and we will focus on a family of recovery algorithms called *ARIES (Algorithms for Recovery and Isolation Exploiting Semantics). The ultimate goal of this portion of the material is to provide us with a clear conceptual framework to reflect about recovery strategies, and to predict how different recovery strategies behave in different scenarios.*

The learning goals for this portion of the material are listed below.

- Explain the concepts of volatile, nonvolatile, and stable storage as well as the main assumptions underlying database recovery.
- Predict how force/no-force and steal/no-steal strategies for writes and buffer management influence the need for redo and undo.
- Explain the notion of logging and the concept of write-ahead logging.
- Predict what portions of the log and database are necessary for recovery under different failure scenarios.
- Explain how write-ahead logging is achieved in the ARIES protocol.
- Explain the functions of recovery metadata such as the transaction table and the dirty page table.

- Predict how recovery metadata is updated during normal operation.
- Interpret the contents of the log resulting from ARIES normal operation.
- Explain the three phases of ARIES crash recovery: analysis, redo, and undo.
- Predict how recovery metadata, system state, and the log are updated during recovery.

Chapter 6

Experimental Design

This chapter contains the book chapters:

> D. Lilja. Measuring Computer Performance: A Practitioner's Guide. Chapters 1, 2, and 6, pp. 1–24 and 82–107 (50 of 261). Cambridge University Press, 2000. ISBN: 978-0-521-64105-0

Properly comparing different designs and implementations of a given service or system is a hard problem. A wealth of techniques have been developed to measure or estimate the performance of computer systems, including analytical modeling, simulation, and experimentation. Use of these techniques requires care in assumption documentation, metric selection, benchmark design, measurement approach, and experimental setup. While a comprehensive treatment is beyond the scope of ACS, we intend to provide an overview of the issues, as well as measurement exercises to develop competence in structuring experiments. *The ultimate goal of this portion of the material is to provide us with basic concepts in measurement of computer systems, as well as allow us to structure and carry out experiments to measure basic performance properties of a system under study.*

The learning goals for this portion of the material are listed below.

- Explain the three main methodologies for performance measurement and modeling: analytical modeling, simulation, and experimentation.
- Design and execute experiments to measure the performance of a system.

Chapter 6 of Lilja's book is given to deepen understanding of available measurement strategies; however, it is to be considered as an additional reading and not fundamental to the attainment of the learning goals above.

Chapter 7

Notions of Reliability

This chapter contains the book chapter:

> J. H. Saltzer and M. F. Kaashoek. Principles of Computer System Design: An Introduction. Part II. Chapters 8.1–4 and 8.6, pp 8-2 – 8-35 and 8-51 – 8-54 (38 of 826). Creative Commons License, 2009.

An important challenge in providing **high availability** is architecting computer systems for reliability. The first aspect to be studied refers to characterizing basic reliability concepts, and estimating quantitative measures of reliability of given system configurations. Many of these quantitative estimates are based on underlying statistical assumptions, often of independent random failures. It is important to discuss the relationship between these assumptions and real-world behavior, so that the adequate scope and use of reliability metrics can be determined. A second aspect to be studied are strategies and techniques for fault-tolerance. Many strategies are based on the idea of selectively introducing redundancy in data and components. An important class of techniques is the class of replication techniques. When combined with asynchronous communication, replication can be an effective tool to increase availability; however, this benefit is counterbalanced by the impact on the system property of atomicity. *The ultimate goal of this portion of the material is to arm us with the basic methodologies for designing services which include appropriate mechanisms for fault-tolerance, including basic replication protocols, as well as allow us to reflect on the applicability and limitations of the fault-tolerance strategies employed in a given scenario.*

The learning goals for this portion of the material are listed below.

- Identify hardware scale of highly-available services common by current standards.
- Predict reliability of a component configuration based on metrics such as failure probability, MTTF/MTTR/MTBF, availability/downtime.
- Explain how assumptions of reliability models are at odds with observed events.
- Explain and apply common fault-tolerance strategies such as error detection, containment, and masking.

- Explain techniques for redundancy, such as n-version programming, error coding, duplicated components, replication.
- Categorize main variants of replication techniques and implement simple replication protocols.

This page has intentionally been left blank.

Chapter 8

Topics in Distributed Coordination and Distributed Transactions

This chapter contains parts of the book chapter:

> G. Couloris, J. Dollimore, T. Kindberg. Distributed systems, concepts and design. Third Edition. Chapters 11 (except 11.2 and 11.3) and 13, pp. 419–423, 436–464, and 515–552 (72 of 772). Addison-Wesley ,2001. ISBN: 0201-61918-0

In addition, the chapter also contains the paper:

> F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22(4) pp. 299–319 (21 of 409), 1990. Doi: 10.1145/98163.98167

Achieving both **high availability** and **atomicity** simultaneously is a hard problem. As we have seen previously, two strategies employed for availability are redundancy via replication and distribution of system functionality. When these techniques are employed, atomicity of our higher-level service is typically compromised. An important question is whether atomicity can be reestablished even when replication or distribution are employed. The text explores techniques from distributed systems to provide us with an only partially positive answer, which depends fundamentally on the assumptions made about the underlying failure model. First, the text explores the primitive of multicast communication, which can be used as a building block in a replication protocol. Second, we turn to distributed transactions, which can be used to atomically interact with a system with distributed functionality. *The ultimate goal of this portion of the material is to provides us with an overview of the results in distributed systems related to synchronous replication and distributed transactions, as well as basic protocols to achieve both, in particular totally-ordered multicast and two-phase commit.*

The learning goals for this portion of the material are listed below.

- Explain the difficulties of guaranteeing atomicity in a replicated distributed system.

- Explain the notion of state-machine replication and the ISIS approach to totally ordered multicast among replicas.
- Describe the implications of the FLP impossibility result and possible workarounds.
- Explain mechanisms necessary for distributed transactions, such as distributed locking and distributed recovery.
- Explain the operation of the two-phase commit protocol (2PC).
- Predict outcomes of 2PC under failure scenarios.

Schneider's tutorial paper on state-machine replication is given to deepen understanding of this approach to replication; however, it is to be considered as an additional reading and not fundamental to the attainment of the learning goals above.

By contrast, particular attention should be paid above to the material on totally-ordered multicast (Section 11.4.3) and the overview of the consensus problem and its related theoretical results (Sections 11.5.1, 11.5.2, and 11.5.4), as well as to the material on distributed transactions, especially the two-phase commit protocol (Couloris et al.'s Chapter 13, especially Section 13.3).

Chapter 9

Communication and End-to-End Argument

This chapter contains parts of the book chapter:

> A. S. Tanenbaum and M. V. Steen. Distributed Systems: Principles and Paradigms. Second Edition. Chapters 4, pp. 124–125 and 140–177 (40 of 686). Pearson International Edition, 2007. ISBN: 0-13-613553-6

In addition, the chapter includes the papers:

> D. Pritchett. BASE: An ACID Alternative. Queue 6, 3 (May 2008), pp. 48-55 (8 of 72), 2008. Doi=10.1145/1394127.1394128.

> J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. ACM Trans. Comput. Syst. 2(4) pp. 277–288 (12 of 359), 1984. Doi: 10.1145/357401.357402

In this chapter, we observe that an important programming tactic to obtain availability is to decouple system components through *asynchronous communication abstractions*. The text reviews communication abstractions that follow this model, with particular emphasis on Message-Oriented Middleware (MOM). In addition, the text also discusses other important such communication abstractions, such as streaming communication as well as gossiping. Armed with this background, we turn to papers which discuss how to architect applications with such abstractions. First, the BASE (basically available, soft state, eventually consistent) methodology is introduced, which argues for decoupling system functions by use of reliable MOM intermediaries. Second, the end-to-end argument is introduced, which debates at a higher level which functionality and guarantees a system must actually provide, and which functionality can be left to applications. *The ultimate goal of this portion of the material is to encourage us to reflect on the relationship between application and system architecture, and how the use of different system abstractions my affect high-level properties such as availability.*

The learning goals for this portion of the material are listed below.

- Describe different approaches to design communication abstractions, e.g., transient vs. persistent and synchronous vs. asynchronous.

- Explain the design and implementation of message-oriented middleware (MOM).
- Discuss alternative communication abstractions such as data streams and multicast/gossip.
- Explain how to organize systems employing a BASE methodology.
- Discuss the relationship of BASE to eventual consistency and the CAP theorem.
- Apply the end-to-end argument to system design situations.

This page has intentionally been left blank.

This page has intentionally been left blank.

This page has intentionally been left blank.

Chapter 10

Data Processing - External Sorting

This chapter contains the book chapter:

> H. Garcia-Molina, J. D. Ullman, J. Widom. Database Systems: The Complete Book. Chapter 11.4, pp. 525–533 (9 of 1119). Prentice Hall, 2002. ISBN: 0-13-031995-3

A vast number of applications must process and analyze large quantities of data. These applications need the system property of **scalability** with data volumes. Classic algorithms are developed with the RAM model in mind, and often break or perform very poorly over multi-level memories. This necessitates the study of algorithms under different models. In the following, the text studies algorithms for data processing under the external memory model. In this chapter, the ubiquitous problem of *sorting* is studied. *The ultimate goal of this portion of the material is to allow us to reflect on how the external memory model differs from the classic model of algorithm design, and to explore in detail the example of sorting as a first data processing service implemented under the external memory model.*

The learning goals for this portion of the material are listed below.

- Identify the problem of processing large data collections with external memory algorithms.
- Discuss the implications of the external memory model on algorithm design.
- Explain the classic sort-merge external sorting algorithm, and in particular its two-phase, multiway variants.

This page has intentionally been left blank.

Chapter 11

Data Processing - Basic Relational Operators and Joins

This chapter contains the book chapter:

> H. Garcia-Molina, J. D. Ullman, J. Widom. Database Systems: The Complete Book. Chapter 15, pp. 713–774 (62 of 1119). Prentice Hall, 2002. ISBN: 0-13-031995-3

Other than sorting, a set of operations commonly applied to bulk data sets are the operations of the relational algebra. This chapter discusses how to design algorithms for and implement these operators under the external memory model, maintaining the property of **scalability** with data volumes. *The ultimate goal of this portion of the material is to equip us with a variety of algorithmic design alternatives for services which must analyze large data sets, in particular memory management approaches, indexing, sorting, and hashing.*

The learning goals for this portion of the material are listed below.

- Explain the design of operators for basic relational operations, including their scan-based, one-pass variants as well as associated memory management issues.
- Analyze algorithms for the join operation, including the multiple variants based on nested loops, indexing, sorting, and hashing as well as one, two, and many passes.
- Apply external memory algorithms to the implementation of data processing operators.

This page has intentionally been left blank.

Chapter 12

Data Processing - Parallelism

This chapter contains the papers:

D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. Commun. ACM 35, pp. 85-98 (14 of 1868), 1992. Doi: 10.1145/129888.129894

J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. Commun. ACM 53, 1, pp. 72-77 (6 of 159), 2010. Doi: 10.1145/1629175.1629198

G. Graefe. Encapsulation of parallelism in the Volcano query processing system. SIGMOD Rec. 19, 2, pp. 102-111 (10 of 632), 1990. Doi: 10.1145/93605.98720

An important technique for **scalability** with data volumes is the use of *parallelism.* Parallel data processing methodologies allow us to add more processing and data transfer components to a computer system, with the goal to either obtain speedups or to scale up with data. In this final chapter, the text explores basic concepts in parallel data processing. *The ultimate goal of this portion of the material is to provide us with the basic terminology and approaches to parallel processing of large data volumes, which can be employed to reflect on the design and implementation of data analysis services.*

The learning goals for this portion of the material are listed below.

- Identify the main metrics in parallel data processing, namely speed-up and scale-up.
- Describe different models of parallelism (partition, pipelined) and architectures for parallel data processing (shared-memory, shared-disk, shared-nothing).
- Explain different data partitioning strategies as well as their advantages and disadvantages.
- Apply data partitioning to achieve parallelism in data processing operators.
- Explain the main algorithms for parallel processing, in particular parallel scan, parallel sort, and parallel joins.

- Explain the relationship between MapReduce and partitioned parallel processing strategies.

Graefe's paper on interface design for parallel operators is given to deepen understanding; however, it is to be considered as an additional reading and not fundamental to the attainment of the learning goals above.