159.355 Concurrent Systems

Hans W. Guesgen

Based on slides provided with:

Mordechai (Moti) Ben-Ari
Principles of Concurrent and Distributed Programming (SE)
Addison-Wesley, 2006

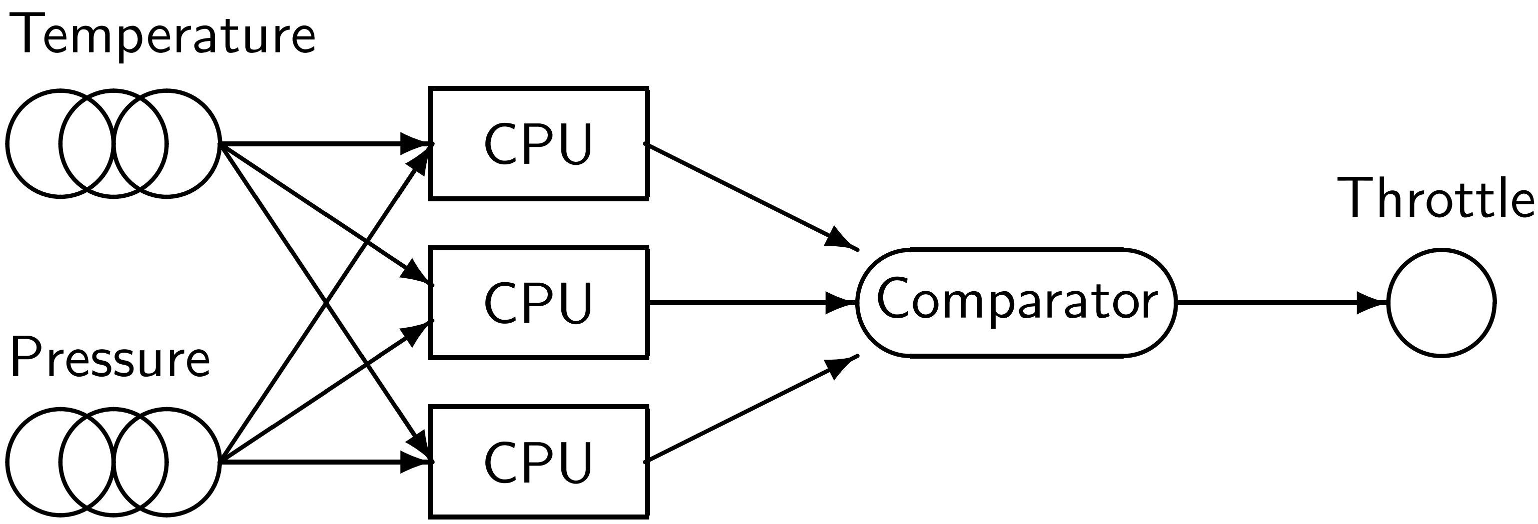
`http://www.weizmann.ac.il/sci-tea/benari/`

Consensus

- One of the primary motivations for building distributed systems is to improve reliability.
- There are two properties to achieve in a reliable system
  - **Fail safe**
  - System failures do not cause damage to the system or to its users.

  - **Fault tolerant**
  - System continues to fulfil its requirements even if there are failures.
- A distributed system is not automatically fail safe or fault tolerant.

Architecture for a Reliable System



- When input sensors are replicated, they may not all give exactly the same data.
- A faulty input sensor or processor may not fail gracefully.
- If all processors are using the same software, the system is not tolerant of software bugs.

The Problem Statement

- A group of Byzantine armies is surrounding an enemy city.
- If all armies attack together, they can capture the city.
- Otherwise, they must all retreat to avoid defeat.
- The generals have reliable messengers but some of the generals may be traitors.
- The task is to devise an algorithm so that all loyal generals come to a consensus.

- Problem statement applied to faults in distributed systems:
  - **Crash failures**
  - A failure node (traitor) stops sending messages at any arbitrary point during the execution of the algorithm.

  - **Byzantine failures**
  - A traitor can send arbitrary messages, not just the messages required by the algorithm.

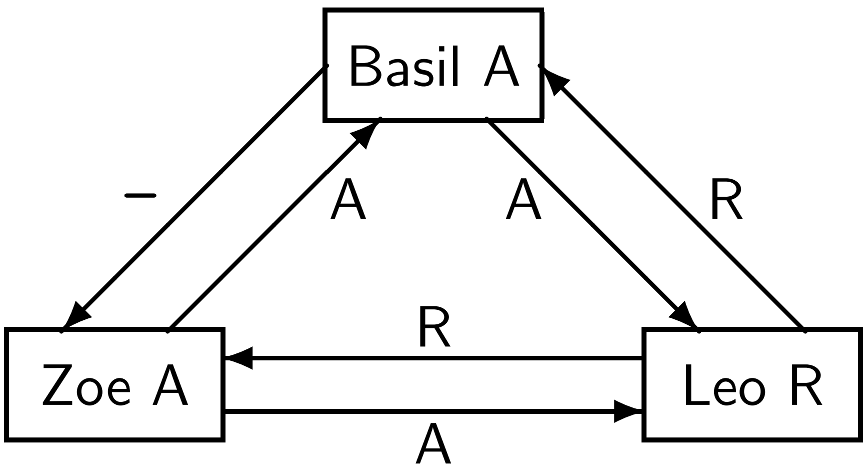
Algorithm 12.1: Consensus - one-round algorithm

```
        planType finalPlan
        planType array[generals] plan

p1: plan[myID] ← chooseAttackOrRetreat
p2: for all other generals G
p3:     send(G, myID, plan[myID])
p4: for all other generals G
p5:     receive(G, plan[G])
p6: finalPlan ← majority(plan)
```

- The values of planType are A for attack and R for retreat.
- Each general chooses a plan and sends it to the other generals.
- The final plan is the majority vote among all plans
  (both the general's own plan and the plans received from the others).

Messages Sent in a One-Round Algorithm



- Suppose Zoe and Leo are loyal, while Basil is a traitor.
- Basil and Zoe choose to attack, while Leo chooses to retreat.
- Basil crashes after sending a message to Leo and before sending a similar message to Zoe.

Data Structures in a One-Round Algorithm

Leo	
general	plan
Basil	A
Leo	R
Zoe	A
majority	A

Zoe	
general	plans
Basil	–
Leo	R
Zoe	A
majority	R

- By a majority vote of 2–1, Leo chooses A.
- Zoe chooses R, because we assume that ties are broken in favour of R.
- If a general crashes, the remaining loyal generals can fail to come to a consensus.

The Byzantine Generals Algorithm

- The one-round algorithm does not use the fact that certain generals are loyal.
- An individual node cannot know the identities of the traitors directly but must ensure that the plan of the traitors cannot prevent the loyal generals from reaching consensus.
- The Byzantine Generals algorithm achieves this by using extra rounds of sending messages:
  - In the first round, each general sends its own plan.
  - In subsequent rounds, each general sends what it received from other generals about their plans.
- By definition, loyal generals always relay exactly what they received.
- If there are enough loyal generals, they can overcome the attempts of the traitors to prevent them from reaching a consensus.

Algorithm 12.2: Consensus - Byzantine Generals algorithm

```
planType finalPlan
planType array[generals] plan, majorityPlan
planType array[generals, generals] reportedPlan

p1:  plan[myID] ← chooseAttackOrRetreat
p2:  for all other generals G                       // First round
p3:     send(G, myID, plan[myID])
p4:  for all other generals G
p5:     receive(G, plan[G])
p6:  for all other generals G                       // Second round
p7:     for all other  generals G' except G
p8:        send(G', myID, G, plan[G])
p9:  for all other generals G
p10:    for all other generals G' except G
p11:       receive(G, G', reportedPlan[G, G'])
p12: for all other generals G                       // First vote
p13:    majorityPlan[G] ← majority(plan[G] ∪ reportedPlan[*, G])
p14: majorityPlan[myID] ← plan[myID]               // Second vote
p15: finalPlan ← majority(majorityPlan)
```

Crash Failure - First Scenario (Leo)

Leo				
general	plan	reported by		majority
		Basil	Zoe	
Basil	A		–	A
Leo	R			R
Zoe	A	–		A
majority				A

- Suppose Zoe and Leo are loyal, while Basil is a traitor.
- Basil and Zoe choose to attack, while Leo chooses to retreat.
- Basil sends a message to Leo before crashing.

Crash Failure - First Scenario (Zoe)

Zoe				
general	plan	reported by		majority
		Basil	Leo	
Basil	–		**A**	A
Leo	R	–		R
Zoe	A			A
majority				A

- Basil crashes before sending a message to Zoe.
- Leo sends the plan to retreat to Zoe in the first round.
- Leo relays Basil's plan to Zoe in the second round.

Crash Failure - Second Scenario (Leo)

Leo				
general	plan	reported by		majority
		Basil	Zoe	
Basil	A		A	A
Leo	R			R
Zoe	A	A		A
majority				A

- Both Basil and Zoe report their plans to Leo in the first round.
- Both Basil and Zoe relay the other general's plan to Leo in the second round.

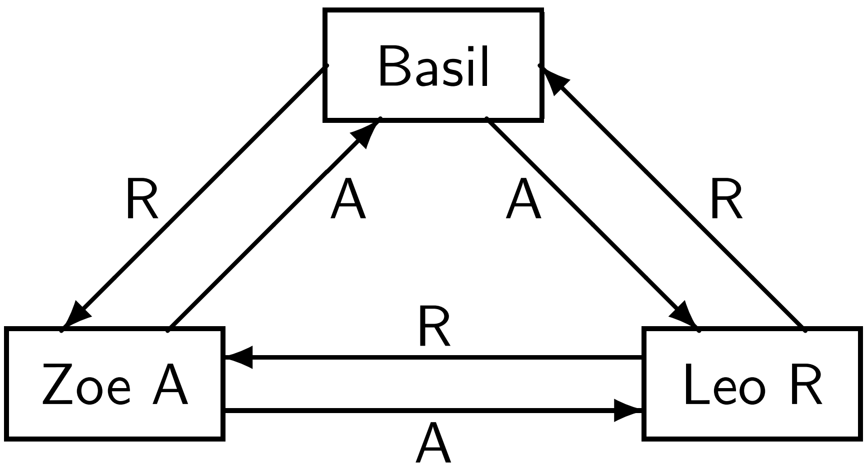
Crash Failure - Second Scenario (Zoe)

Zoe				
general	plan	reported by		majority
		Basil	Leo	
Basil	A		A	A
Leo	R	–		R
Zoe	A			A
majority				A

- Leo sends the plan to retreat to Zoe in the first round.
- Leo relays Basil's plan to Zoe in the second round.
- Basil sends the plan to attack to Zoe in the first round but crashes before relaying Leo's plan.

- In both scenarios, the loyal generals have consistent data structures and come to the same decision about the final plan!

Byzantine Failure with Three Generals



- Suppose Zoe and Leo are loyal, while Basil is a traitor.
- A traitor is allowed to send an attack or retreat message, regardless of its internal state.
- Basil sends an attack message to Leo but a retreat message to Zoe.

Data Stuctures for Leo and Zoe After First Round

Leo	
general	plans
Basil	A
Leo	R
Zoe	A
majority	A

Zoe	
general	plans
Basil	R
Leo	R
Zoe	A
majority	R

- After the first round, Leo and Zoe reach different decisions.
- No surprise, because the one-round algorithm was not correct even in the presence of crash failures.

Data Stuctures for Leo After Second Round

Leo				
general	plans	reported by		majority
		Basil	Zoe	
Basil	A		A	A
Leo	R			R
Zoe	A	**R**		R
majority				R

- In the second round, Basil erroneously reports to Leo that Zoe's plan is to retreat.
- This causes Leo to make an erroneous decision about Zoe's plan (tie-break).

Data Stuctures for Zoe After Second Round

Zoe				
general	plans	reported by		majority
		Basil	Leo	
Basil	A		A	A
Leo	R	R		R
Zoe	A			A
majority				A

- In the second round, Basil correctly reports to Zoe that Leo's plan is to retreat.
- The two loyal generals reach inconsistent final decisions.
- This means that the algorithm is incorrect for three generals of whom one is a traitor.

Four Generals: Data Structure of Basil (1)

Basil					
general	plan	reported by			majority
		John	Leo	Zoe	
Basil	A				A
John	**A**		**A**	?	A
Leo	R	R		?	R
Zoe	?	?	?		?
majority					?

- Suppose Basil, John, and Leo are loyal, while Zoe is a traitor.
- Basil and John choose to attack, while Leo chooses to retreat.
- Basil receives the correct plan of John, both directly from John as well as indirectly from Leo.
- The report from Zoe cannot change the majority vote for John.
- The same holds for Leo.

Four Generals: Data Structure of Basil (2)

Basil					
general	plans	reported by			majority
		John	Leo	Zoe	
Basil	A				A
John	A		A	?	A
Leo	R	R		?	R
Zoe	**R**	**A**	**R**		R
					R

- Suppose Zoe sends first-round retreat messages to Basil and Leo, but an attack message to John.
- These are relayed correctly in the second round by the loyal generals.
- Regardless of what messages Zoe sends, the loyal generals come to the same decision about Zoe's plan.

Complexity of the Byzantine Generals Algorithm

traitors	generals	messages
1	4	36
2	7	392
3	10	1790
4	13	5408

- The Byzantine Generals algorithm can be generalised to any number of generals.
- For every additional traitor, an additional round of messages must be sent.
- The total number of generals must be at least $3t + 1$, where t is the number of traitors.

- The algorithm quickly becomes impractical as the number of traitors increases!

Algorithm 12.3: Consensus - flooding algorithm
planType finalPlan set of planType plan $\leftarrow$ { chooseAttackOrRetreat } set of planType receivedPlan
p1: do $t+1$ times p2:   for all *other* generals G p3:   send(G, plan) p4:   for all *other* generals G p5:   receive(G, receivedPlan) p6:   plan $\leftarrow$ plan $\cup$ receivedPlan p7: finalPlan $\leftarrow$ majority(plan)

- Very simple algorithm for consensus in the presence of crash failures.
- Each general repeatedly sends the set of plans that he has received.
- It is sufficient that a single such message from a loyal general reaches every other loyal general.
- If there are t traitors and $t+1$ rounds of sending and receiving messages, then one such message must have been sent and received without crashing.

The King Algorithm

- The Byzantine Generals algorithm requires a large number of messages, while the King algorithm gets away with fewer messages:

Byzantine Generals		
traitors	generals	messages
1	4	36
2	7	392
3	10	1790
4	13	5408

King		
traitors	generals	messages
1	5	48
2	9	240
3	13	672
4	17	1440

- The downside is that an extra general is required per traitor, which means that the total number of generals must be at least $4t + 1$, where t is the number of traitors.
- The idea of the algorithm is to give one general in each round the special status of king.
- The king sends his plan to the other generals, who consider replacing their plans with the king's plan.

Algorithm 12.4: Consensus - King algorithm - 5 generals

```
            planType finalPlan, myMajority, kingPlan
            planType array[generals] plan
            integer votesMajority
p1: plan[myID] ← chooseAttackOrRetreat

p2: do two times
p3:     for all other generals G          // First and third rounds
p4:         send(G, myID, plan[myID])
p5:     for all other generals G
p6:         receive(G, plan[G])
p7:     myMajority ← majority(plan)
p8:     votesMajority ← number of votes for myMajority
```

Algorithm 12.4: Consensus - King algorithm - 5 generals (continued)

```
p9:   if my turn to be king                // Second and fourth rounds
p10:     for all other generals G
p11:        send(G, myID, myMajority)
p12:     plan[myID] ← myMajority
      else
p13:     receive(kingID, kingPlan)
p14:     if votesMajority > 3
p15:        plan[myID] ← myMajority
         else
p16:        plan[myID] ← kingPlan

p17: finalPlan ← plan[myID]                // Final decision
```

Scenario for King Algorithm: First King Loyal General Zoe (1)

Basil							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
A	A	R	R	R	R	3	

John							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
A	A	R	A	R	A	3	

Leo							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
A	A	R	A	R	A	3	

Zoe							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
A	A	R	R	R	R	3	

Scenario for King Algorithm: First King Loyal General Zoe (2)

Basil							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R							R

John							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
	R						R

Leo							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
		R					R

Zoe							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
				R			

Scenario for King Algorithm: First King Loyal General Zoe (3)

Basil							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

John							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

Leo							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

Zoe							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

Scenario for King Algorithm: First King Traitor Mike (1)

Basil							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R							R

John							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
	A						A

Leo							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
		A					A

Zoe							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
				R			R

Scenario for King Algorithm:
First King Traitor Mike (2)

Basil							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

John							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

Leo							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

Zoe							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

Scenario for King Algorithm: First King Traitor Mike (3)

Basil							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
A							A

John							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
	A						A

Leo							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
		A					A

Zoe							
Basil	John	Leo	Mike	Zoe	myMajority	votesMajority	kingPlan
				A			