Brief Announcement: Object Oriented Consensus

Yehuda Afek
School of Computer Science, Tel-Aviv University
afek@tau.ac.il

Edo Cohen
School of Computer Science, Tel-Aviv University
edocohen@tau.ac.il

James Aspnes
School of Computer Science, Yale University
james.aspnes@gmail.com

Danny Vainstein
School of Computer Science, Tel-Aviv University
danvainstein@tau.ac.il

ABSTRACT
We suggest a template that reveals the structure of many consensus algorithms as a generic procedure. The template builds on a new object, vacillate-adopt-commit which is an extension of the well known adopt-commit object. In addition we extend Aspnes’s conciliator object to a new object that we call a reconciliator. The consensus algorithm template works in rounds of alternating vacillate-adopt-commit and reconciliator operations. The vacillate-adopt-commit object observes the processors’ preferences and suggests a preference output with a measure of confidence (vacillate, adopt or commit) on the preference. The reconciliator ensures termination, by providing new preferences for the processors. We show how several key consensus algorithms exactly fit our template. Here we demonstrate the decomposition of Ben-Or’s randomized algorithm. The decomposition of the Phase King Byzantine and the Paxos algorithm are given in the full paper [1]. We analyze and compare our template based on vacillate-adopt-commit and reconciliator objects to previous work [3, 5], suggesting a decomposition of consensus based on adopt-commit and conciliator objects. We claim that the three return values of vacillate-adopt-commit more accurately describe existing algorithms.

KEYWORDS
Distributed Algorithms; Consensus; Adopt-Commit;

1 INTRODUCTION
The consensus problem, introduced by Lamport, Pease and Shostak [6] resides at the heart of many distributed algorithms such as leader election, database transaction handling, resource allocation, ensuring storage replicas are mutually consistent and many more.

In the consensus protocol between n processors, each with an input value, processors agree on a single common output which was the input to one of them. While consensus is trivial in a non-faulty synchronous environment, it is often more difficult in practice as most distributed networks are asynchronous and must be resilient to faults of various types.

Gafni [5] proposed the adopt-commit object, an object fulfilling weaker guarantees than consensus, as a building block of consensus. Aspnes [3] further provided a detailed decomposition of consensus into adopt-commit and a complementary conciliator object which together form a generic framework describing consensus. In this work we extend these decompositions, into vacillate-adopt-commit and reconciliator objects, which describe the core of many existing consensus algorithms more accurately than the previous decomposition.

The paper is structured as follows. In Section 3, the consensus template is presented. In Section 4, we demonstrate how the decomposition applies to Ben-Or’s consensus algorithm (in our full paper [1] we further apply our decomposition to the Paxos and Phase-King consensus algorithms). Section 5 explores the relation between vacillate-adopt-commit and adopt-commit, showing the latter is slightly weaker.

2 PRELIMINARIES
Consensus guarantees the following three properties: Validity - The output value, u, was provided as an input to the object by some processor. Termination - The output is returned within a finite number of steps. Agreement - All processors invoking the object receive the same output value u.

Adopt-commit is a weaker form of consensus which guarantees validity and termination. The object supports a single operation, adopt-commit(v) and returns some preference u with a level of confidence, adopt or commit. The object does not guarantee agreement, instead, adopt-commit holds the following guarantees: Coherence - If some processor receives (commit, v), any other returned value (with either commit or adopt) must have the same value v. Convergence - If all processors invoke adopt-commit(v) with the same value v, the object must return (commit, v) with the same value v to all processors.

Vacillate-adopt-commit is an extension of adopt-commit which supports a single operation, vacillate-adopt-commit(v) and returns some preference u and a level of confidence, vacillate, adopt or commit. As opposed to adopt-commit, vacillate-adopt-commit adds a level of confidence and guarantees validity, termination and convergence. Coherence is modified to suit three levels of confidence, therefore, vacillate-adopt-commit guarantees coherence over adopt & commit - If any processor receives (commit, v), then any other returned value is either (commit, v) or (adopt, v) (with the same value v). Coherence over vacillate & adopt - If no processor received commit and some processor received (adopt, u), then every other processor receives either (adopt, u) or (vacillate, w) where w may be any value.
The **reconciliator** object also supports a single operation, **reconciliator**(v) and only returns a preference u. The object guarantees that every value has a non-zero probability (that may depend on n, the number of processors) to be returned to all processors. In the Byzantine setting, the **reconciliator** guarantees to return the same value to all processors eventually (i.e., if called sufficiently many times). 1

### 3 THE GENERIC FORM OF CONSENSUS

The thesis of this paper is that many well known consensus algorithms have the same basic structure consisting of two objects, VAC 2 that checks whether consensus has been reached or not and **reconciliator** that shakes up the preferences of the processors in case of a stalemate. We do not claim that this structure is necessarily better than using AC and **conciliators**, but that it more accurately reflects the existing structure of algorithms in the literature.

Informally, the generic consensus algorithms work in rounds. In each round, first the VAC is invoked to observe the system state and tell whether consensus has been reached. VAC returns to each processor one of three possible outputs: (1) **commit**, v which indicates that the system has reached an agreement on value v, (2) **adopt**, v which indicates that it is possible that some processors in the system have agreed on the value v, and (3) **vacillate**, v indicating that the system is in an indecisive state.

If a processor receives **commit**, v, it is guaranteed that no other processor receives a **vacillate** value and all outputs return with the same value, v. A processor receiving **adopt**, v is guaranteed that any other processor either received a **vacillate** value or received the same preference. Finally, if a processor receives **vacillate**, v, the only guarantee it has is that no other processor received a **commit** value.

We note the key difference between the VAC and AC objects. An adopt-commit object always returns a new value to be adopted by a process, but this is not consistent with the structure of many consensus protocols in the literature. Adding a third option, i.e., **vacillate**, accounts for situations where the algorithm does not force a process to update its preference.

The question is how termination of the consensus can be guaranteed if the collection of preferences is balanced and the VAC continually returns vacillate. For that purpose, the **reconciliator** is used to give each vacillating processor a new preference with a guarantee to provide a deciding set of preferences with some probability. That is, whenever a processor receives a **vacillate** value from the VAC object, the distributed **reconciliator** provides each processor with an alternate preference such that eventually enough processors will get the same preference leading to VAC eventually observing agreement. Pseudocode for the consensus template is given in Algorithm 1.

Algorithm 1: Consensus Template

```plaintext
1: Consensus(v)
2: m ← 0
3: INIT()
4: while true do
5:   m ← m + 1
6:   (X, σ) ← VAC(v, m)
7:   σ′ ← reconciliator(σ, m)
8:   switch X do
9:     case vacillate: v ← σ'
10:    case adopt: v ← σ
11:    case commit: v ← σ and decide σ
12: end
13: end
```

Algorithm 2: Ben-Or’s **reconciliator** implementation

```plaintext
1: Reconciliator(X, σ, m)
2: return CoinFlip()
```

Lemma 3.1. Algorithm 1 is a correct consensus algorithm.

Proof. Agreement and validity follow from VAC’s coherence and validity respectively. Termination follows from the convergence property of the **reconciliator** object.

### 4 DECOMPOSING BEN-OR’S ALGORITHM

In this section we show how Ben-Or’s algorithm [4] can be described using our consensus template. Throughout this section the settings are asynchronous, message-passing model and the number of tolerated crash failures, t, is strictly smaller than n/2. Algorithms 2 and 3 are the VAC’s and **reconciliator**’s implementations, Lemmas 4.1 and 4.2 prove the implementations correctness, i.e., that they uphold the objects’ guarantees.

We note the key difference between the VAC and AC objects. An adopt-commit object always returns a new value to be adopted by a process, but this is not consistent with the structure of many consensus protocols in the literature. Adding a third option, i.e., **vacillate**, accounts for situations where the algorithm does not force a process to update its preference.

The question is how termination of the consensus can be guaranteed if the collection of preferences is balanced and the VAC continually returns vacillate. For that purpose, the **reconciliator** is used to give each vacillating processor a new preference with a guarantee to provide a deciding set of preferences with some probability. That is, whenever a processor receives a **vacillate** value from the VAC object, the distributed **reconciliator** provides each processor with an alternate preference such that eventually enough processors will get the same preference leading to VAC eventually observing agreement. Pseudocode for the consensus template is given in Algorithm 1.

Note that INIT is a void function unless stated otherwise. Furthermore, note that the operation, decide σ, is followed by a halt operation, that is, the processor will decide upon its value and return. The argument m is the phase of the consensus process.

Next we prove that the template indeed achieves consensus, using the VAC and **reconciliator** properties.

**Lemma 3.1.** Algorithm 1 is a correct consensus algorithm.

**Proof.** Agreement and validity follow from VAC’s coherence and validity respectively. Termination follows from the convergence property of the **reconciliator** object.

**Lemma 4.1.** Algorithm 2 is a correct **reconciliator** implementation.

**Proof.** **Probabilistic convergence:** For any given value v, \( P(\text{all processors finish with } v \text{ after the coin flip}) \geq \frac{1}{2^n} > 0 \)

**Lemma 4.2.** Algorithm 3 is a correct **vacillate-adopt-commit** implementation.

**Proof.** The proof is similar to the Ben-Or algorithm correctness proof found in the survey of Aspnes [2].

**Validity, termination and convergence** follow easily from the implementation.

**Coherence over commit and adopt,** assume a process received (commit, v). Therefore, it received more than \( t \) **ratify** messages, meaning a non-faulty processor sent out a **ratify** message to all
Differentiation

Algorithm 3: Ben-Or’s vacillate-adopt-commit implementation

other processors - therefore it is enough to show that every 2 vacillate messages have the same value, v (since then all processes received at least one vacillate message with the same value, v). Assume towards contradiction that this isn’t the case - meaning messages (2, u, vacillate) and (2, u, vacillate) where u ≠ v had been sent. By the first if statement, this means there’s a processor that sent out both u and v which is a contradiction.

Coherence over adopt and vacillate. since every 2 vacillate messages have the same value, v, coherence follows.

5 VACCILLATE-ADOPT-COMMIT VS ADOPT-COMMIT

5.1 ADOPT-COMMIT Is Not Enough

The concept of decomposing consensus into separate objects is by no means original and was formally presented in [5]. Later work by Aspnes [3] described a framework of adopt-commit objects that detect agreement, and conciliators that ensure agreement with some probability. We argue that this decomposition breaches abstraction responsibilities. The consensus algorithm may be viewed as an iterative process which begins in some arbitrary state and terminates with an agreement upon a valid value. Under this interpretation, the framework of repetitive adopt-commit followed by conciliator fails to distinguish between the phases of the consensus procedure in which part of the participants have terminated and others have not. Due to this limitation and the coherence constraint, the conciliator must be aware of the global state of the consensus procedure in order to ensure validity.

In order to make our argument more concrete, we demonstrate how Ben-Or’s consensus algorithm cannot be described as a sequence of repetitive vacillate-adopt-commit followed by conciliator.

To demonstrate the problem with formulating Ben-Or’s consensus protocol using a series of adopt-commit and conciliator objects, consider each round of Ben-Or’s algorithm [4]. Let P be a processor participating in the agreement process. P experiences one of three possible outcomes: (1) not receiving any ratify message. (2) receiving up to t ratify messages. (3) receiving more than t ratify messages.

These outcomes correspond to vacillate, adopt, and commit, respectively. Option 1 fits a processor which received vacillate as it has no guarantees about other values received by other processors. Option 2 corresponds to adopt under the VAC framework, since by coherence, any processor that received (adopt, v) is guaranteed that every other processor that received either vacillate or commit, also received the value v. Option 3 corresponds to commit, since any processor that received (commit, v) is guaranteed that all other processors received either (commit, v) or (adopt, v). However, using only adopt-commit objects is not enough in order to describe these three options.

It might be tempting to assume that two consecutive adopt-commit objects might resolve this entanglement as we have shown that VAC may be implemented using two AC objects. We argue this is not the case, that is, we claim that the sequence of U = A−1; A0; A1; . . . also fails to describe Ben-Or’s consensus protocol. In order to describe option (2) the first adopt-commit must return adopt while the second returns commit. However, the decomposition framework described in [3] requires that upon reception of commit the processor immediately decides on the value received, whereas it is possible that in Ben-Or’s protocol such a state is reached with value u but a final agreement is achieved with value u′ ≠ u.

5.2 Relation Between Vaccillate-Adopt-Commit and Adopt-Commit

In addition to the former section, we demonstrate in the full version of our paper [1] that one may easily simulate an adopt-commit object using a VAC object. On the other hand, one may successfully simulate VAC using two sequential calls to adopt-commit objects. In our opinion this further shows an inherent difference between the two objects.

REFERENCES


---

\(U = A_{-1}; A_0; C_1; A_1; C_2; A_2; \ldots\)