Resilient Consensus for Infinitely Many Processes

(Extended Abstract)

Michael Merritt[1] and **Gadi Taubenfeld**[2]

[1] AT&T Labs, 180 Park Ave., Florham Park, NJ 07932, USA.
mischu@research.att.com

[2] School of computer science, the Interdisciplinary Center, P.O.Box 167, Herzliya 46150, Israel. tgadi@idc.ac.il

Abstract. We provide results for implementing resilient consensus for a (countably) infinite collection of processes.

- For a known number of faults, we prove the following equivalence result: For every $t \geq 1$, there is a t-resilient consensus object for infinitely many processes if and only if there is a t-resilient consensus object for $t + 1$ processes.
- For an unknown or infinite number of faults, we consider whether an infinite set of wait-free consensus objects, capable of solving consensus for any finite collection of processes, suffice to solve wait-free consensus for infinitely many processes. We show that this implication holds under an assumption precluding runs in which the number of simultaneously active processes is not bounded, leaving the general question open.

All the proofs are constructive and several of the constructions have adaptive time complexity. (Reduced to the finite domain, some improve on the time complexity of known results.) Furthermore, we prove that the constructions are optimal in some space parameters by providing tight simultaneous-access and space lower bounds. Finally, using known techniques, we draw new conclusions on the universality of resilient consensus objects in the infinite domain.

1 Introduction

We explore the solvability of consensus when the number of processes which may participate is countably infinite. The investigation broadens our understanding of the limits of fault-tolerant computation. Recent work has investigated the the design of algorithms assuming no *a priori* bound on the number of processes [ASS02,CM02,GMT01,MT00]. Moreover, these assume that the number of active processes may be infinite (in infinite runs). The primary motivation for such an investigation is to understand the limits of distributed computation. While in practice the number of processes will always be finite, algorithms designed for an infinite number of processes may scale well: their time complexity may depend on the actual contention and not on the total number of processes.

1.1 Basic concepts

A factor in designing algorithms where the number of processes is unknown is the *concurrency level,* the maximum number of processes that may be active simultaneously. (That is, processes participating in the algorithm at the same instant of time. This is often called *point contention.* A weaker possible definition of concurrency, often called *interval contention*, is not considered here.) Following [MT00,GMT01], we distinguish between the following concurrency levels:

- *finite:* There is a finite bound (denoted by c) on the maximum number of processes that are simultaneously active, over all runs.
- *bounded:* In each run, there is a finite bound on the maximum number of processes that are simultaneously active. (But there is no finite bound over all runs.)
- *unbounded:* In each run, the number of processes that are simultaneously active is finite but can grow without bound.

Notice that although an infinite number of processes may take steps in the same run, we assume that the concurrency in any single state is finite.

Time complexity is computed using the standard model, in which each primitive operation on a shared object is assumed to take no more than one time unit. An algorithm is *adaptive to process contention* if the time complexity of processes' operations is bounded by a function of the number of processes active before and concurrently with those operations. It is *adaptive to operation contention* if the time complexity of processes' operations is bounded by a function of the number of operations active before and concurrently with those operations. (The term *contention sensitive* was first used to describe such algorithms [MT93], but later the term *adaptive* become commonly used.)

Each shared object presents a set of operations. For example, $x.\mathsf{op}$ denotes operation op on object x. For each such operation $x.\mathsf{op}$ on x, there is an associated access control list, denoted ACL($x.\mathsf{op}$), which is the set of processes allowed to *invoke* that operation. Each operation execution begins with an invocation by a process in the operation's ACL, and remains pending until a response is received by the invoking process. The ACLs for two different operations on the same object can differ, as can the ACLs for the same operation on two different objects. A process not in the ACL for $x.\mathsf{op}$ cannot invoke $x.\mathsf{op}$.

A process may be either *correct* or *faulty.* Correct processes are constrained to obey their specifications. A faulty processes follows its protocol up to a certain point and then stops. (I.e., no Byzantine faults.) We generally use t to denote the maximum number of faulty processes, and throughout the rest of the paper, we assume that $t \geq 1$. For any object x, we say x is *t-resilient* if any operation invocation when executed by a correct process, eventually completes in any run in which at most t processes fail. An object is *wait-free* if it can tolerate any number of faults.

Next we define some of the objects used in this paper. An *atomic register* x, is a linearizable object with two operations: $x.\mathsf{read}$ and $x.\mathsf{write}(v)$ where $v \neq \perp$. An $x.\mathsf{read}$ that occurs before the first $x.\mathsf{write}()$ returns $\perp$. An $x.\mathsf{read}$ that occurs after

an x.write() returns the value written in the last preceding x.write() operation. (Throughout, atomic registers are assumed to be wait-free, and with no limits on the number of processes that may access them simultaneously.)

A (binary) *consensus object* x, is a linearizable object with one operation: x.propose(v), where $v \in \{0, 1\}$, satisfying: (1) In any run, the x.propose() operation returns the same value, called the *consensus value*, to every correct process that invokes it. (2) In any finite run, if the consensus value is v, then some process invoked x.propose(v).

Many abstract objects support read operations: operations which return information about the state of the object, without constraining its future behavior (c.f. [Her91]). Atomic registers (and some other abstract objects) also support write() operations: operations that do not return a value, and which constrain future object behavior independently of the state in which they are invoked. These operations have long been known to be weak synchronization primitives [LA87,Her91]. Since we are focusing here on strong synchronization such as consensus, we define an operation to be *powerful* if it is neither a read nor a write() operation. for an object (or object type) x, we define $\text{ACL}_{pow}(x)$ to be the union of $\text{ACL}(x.\text{op})$ for all powerful operations x.op of x.

An object specification also constrains another property: the *access* complexity, Access(x), a bound on the number of distinct processes that may invoke powerful operations in any well-formed run. Obviously, $\text{Access}(x) \leq |\text{ACL}_{pow}(x)|$. An object x is *softwired for n processes* if $\text{ACL}_{pow}(x)$ is the set of all processes (which in this paper is infinite) and $\text{Access}(x) = n$. An object x is *hardwired* for n processes if $\text{Access}(x) = |\text{ACL}_{pow}(x)| = n$.

For $u \geq n > t$, we denote by (u, n, t)-cons a consensus object x that is t-resilient, has $\text{Access}(x) = n$, and $|\text{ACL}_{pow}(x)| = u$. That is, x is a t-resilient consensus object in which in any well-formed run, at most n processes taken from a fixed universe of u processes access it. Thus, $(t+1, t+1, t)$-cons is hardwired and wait-free for $t+1$ processes, more generally (n, n, t) is t-resilient and hardwired for n processes, (∞, n, t)-cons is t-resilient and softwired for n processes, (∞, ∞, t)-cons is t-resilient consensus for an infinite number of processes, and (∞, ∞, ∞)-cons is wait-free consensus for an infinite number of processes.

For sets of object types x and y, the notation $x \Rightarrow y$ (or $y \Leftarrow x$) means that it is possible to implement all the objects of type x using any number of objects of type y and atomic registers. The notation $x \Leftrightarrow y$ means that both $x \Rightarrow y$ and $x \Leftarrow y$.

1.2 Summary of results

We show how to implement t-resilient (and wait-free) consensus objects for infinitely many processes from consensus objects for finitely many processes. Furthermore, we provide tight simultaneous-access and space bounds for these implementations. Simultaneous-access measures the maximum number of processes that are allowed to simultaneously invoke operations, other than atomic reads and writes, on the same primitive object.

Number of faults is known. We show that for every $t \geq 1$: there is a t-resilient consensus object for infinitely many processes iff there is a hardwired t-resilient consensus object for $t+1$ processes:

- $\forall t \geq 1 : [(t+1, t+1, t)\text{-cons} \Leftrightarrow (\infty, t+1, t)\text{-cons} \Leftrightarrow (\infty, \infty, t)\text{-cons}]$.

Number of faults is not known. For an unknown or infinite number of faults, we consider whether an infinite set of wait-free consensus objects, capable of solving consensus for any finite collection of processes, suffice to solve wait-free consensus for infinitely many processes. We show that this implication holds under an assumption precluding runs with unbounded concurrency, leaving the general question open:

- $[\forall t \geq 0 : (t+1, t+1, t)\text{-cons}] \Leftrightarrow (\infty, \infty, \infty)\text{-cons}$ in runs with bounded concurrency.

This result, enables to implement wait-free consensus for infinitely many processes (assuming bounded concurrency) from any known solution for wait-free consensus (deterministic or randomized) for only finitely many processes.

A lower bound. We show that,

- any implementation of a t-resilient consensus object for any number (finite or infinite) of processes and $t \geq 1$ must use: for *every* set of processes T where $|T| = t+1$, at least one object on which the $t+1$ processes in T, can *simultaneously* invoke powerful operations.

This result demonstrates the optimality (in terms of the number of strong objects) of our constructions. Finally, using known techniques, we draw new conclusions on the universality of resilient consensus objects in the infinite domain.

1.3 Related work

We mention below previous work that specifically investigates models with infinitely many processes.

Computing with infinitely many processes has previously been investigated in models with communication primitives stronger than read/write or studying problems such as mutual exclusion that do not admit wait-free solution [MT00]. In [GMT01], wait-free computation using only atomic registers is considered. It is shown that bounding concurrency reveals a strict hierarchy of computational models, of which unbounded concurrency is the weakest model. Nevertheless, it is demonstrate that adaptive versions of many interesting problems (collect, snapshot, renaming) are solvable even in the unbounded concurrency model.

Randomized consensus algorithms for infinitely many processes has been explored in [ASS02]. The strongest result is a wait-free randomized algorithm using only atomic registers. Also, it is stated that standard universal constructions based on consensus continue to work with infinitely many processes with only slight modifications. In [CM02], active disk paxos protocol is implemented for

infinitely many processes. The solution facilitates a solution to the consensus problem with an unbounded number of processes. The solution is based on a collection of a finite number of read-modify-write objects with faults, that emulates a new reliable shared memory abstraction called a ranked register.

1.4 Overview of the paper

The next two sections describe constructions of wait-free and t-resilient consensus for infinitely many processes, from objects for finitely many processes. They address softwired and hardwired base objects, and constructions of softwired from hardwired. Interestingly, several of these constructions match the bounds imposed by the results in Section 4. (Indeed, these bounds guided their discovery.) In addition, several of the constructions presented are adaptive to process or operation contention. In two cases, these lead to improvements of known results for finitely many processes, supporting the intuition that algorithms for infinitely many processes will lead to adaptive and efficient algorithms for the finite case, cf.[GMT01]. The third section following presents and discusses lower bounds, and the final section discusses the universality of resilient consensus objects in the infinite domain.

2 A strong equivalence for t-resilience

The major result of this section is a strong equivalence between hardwired t-resilient consensus and t-resilient consensus for an infinite number of processes:

Theorem 1. $\forall t \geq 1 : [(t+1, t+1, t)\text{-}cons \Leftrightarrow (\infty, \infty, t)\text{-}cons]$.

This theorem extends an equivalence result for finitely many processes alluded to by Chandra *et al* [CHJT94]: $\forall n > t \geq 1 : [(t+1, t+1, t)\text{-cons} \Leftrightarrow (n, n, t)\text{-cons}]$. Theorem 1 together with observations that (∞, ∞, t)-cons objects are universal for t-resilient objects (Corollary 4), implies the universality of the (seemingly) restrictive $(t+1, t+1, t)$-cons objects, even for infinitely many processes.

Theorem 1 is a corollary of the major results of the next two subsections: Theorem 2 in the first subsection shows $(\infty, t+1, t)\text{-cons} \Leftrightarrow (\infty, \infty, t)\text{-cons}$, and Theorem 4 in the second subsection shows that $(t+1, t+1, t)\text{-cons} \Leftrightarrow (\infty, t+1, t)\text{-cons}$. (Note that the arrows from right to left are all trivial.)

2.1 Constructing t-resilient consensus from softwired consensus for $t+1$ processes

This subsection presents a construction implementing t-resilient consensus for an infinite number of processes from softwired consensus for $t+1$ processes:

Theorem 2. $\forall t \geq 1 : [(\infty, t+1, t)\text{-}cons \Leftrightarrow (\infty, \infty, t)\text{-}cons]$.

This theorem is a consequence of the two lemmas that follow: The first implements t-resilient consensus using test&sets, the second shows how to implement the test&sets from $(\infty, 2, 1)$-cons. (The latter are trivial to implement from $(\infty, t+1, t)$-cons.)

Implementing (∞, ∞, t)-cons from $(\infty, t+1, t)$-cons and test&sets

Given a single $(\infty, t+1, t)$-cons object, C, the challenge to implement t-resilient consensus, or (∞, ∞, t)-cons, is to reduce the infinite set of potentially active processes to at most $t+1$, each invoking a separate operation instance on the base object. The algorithm below uses a separate test&set primitive for each of the $t+1$ potential invocations on C.

The linearizable test&set primitive supports a single parameter-less operation, *test*. Runs of instances of the this object are *well-formed* iff no two invocations take the same process identifier as argument. The implementation is trivially wait-free, and in failure-free, well-formed runs, the invocation linearized first returns 1, and the remainder return 0. Moreover, once an invocation returns 0, the set of invocations that may return 1 is finite. It follows that in general runs, if an invocation returns 0 and no other invocation returns 1, there is a failed process with an invocation that is pending forever. By (∞, ∞, ∞)-test&set we denote such an object which supports wait-free *test* operation invocations by infinitely many processes, with any number of crash faults.

Lemma 1. *There is an implementation of* (∞, ∞, t)*-cons using one* $(\infty, t+1, t)$*-cons object,* $t+1$ (∞, ∞, ∞)*-test&set objects, and one register.*

Proof. The implementation in Figure 1 uses $t+1$ instances, $E_1, ..., E_{t+1}$, of (∞, ∞, ∞)-test&set objects. Each instance E_j is used to select a process to access the softwired $(\infty, t+1, t)$-cons object C. Processes move through these object instances in order, each process i invoking $\texttt{test}_i$ on E_1, and moving on to E_2,... if it loses. (That is, if the invocation returns 0.) A process that loses in all $t+1$ test&set objects knows that there are $t+1$ non-empty, disjoint sets of processes contending for those objects. At least one such set contains only correct processes, so it is safe to wait for the consensus value (from C) to be announced via the *Announce* register. Hence, all correct processes either win a test&set and access C, or read the consensus value in *Announce*. □

Reduced to the finite case, the construction in Figure 1 is very similar to a simulation construction in Chandra, *et al* [CHJT94]. The focus of Chandra *et al* is for a model in which non-faulty processes *must* participate, a model we define in Section 4 as *participation required*. This assumption introduces considerable complexity that is the major focus of that paper. Figure 1 indicates that the complex construction and special assumptions needed for the simulation can be greatly simplified when participation is not required. (In the case that participation is required, the relationship between wait-free and t-resilient consensus for infinitely many processes remains essentially unexplored. Given the anomalies and complexities of this model for finitely many proceses (such as the special case t=1, [LH00]), there may be surprises here.)

Implementing test&set objects from $(\infty, 2, 1)$-cons

Lemma 2. (∞, ∞, ∞)*-test&set can be implemented from a register and infinitely many* $(\infty, 2, 1)$*-cons objects.*

```
propose_i(u: boolean), returns boolean                    /* Code of invocation i. */
Shared:
    E_1..E_{t+1}: (∞, ∞, ∞)-test&set objects.
    C: (∞, t + 1, t)-cons.
    Announce: register, initially ⊥.
Local:
    level: integer, initially 1.

1   while (level ≤ t + 1) do
2       if invoke(test, i, E_level) then                  /* If won E_level, */
3           Announce := invoke(propose,u,C)               /* propose u to C, */
4           return(Announce)                              /* set Announce and return, */
5       else level := level + 1                           /* else move to next level. */
        fi
    od                                                    /* If lost all t + 1 levels, */
6   while (Announce = ⊥) do skip od                       /* spin on Announce and return. */
7   return(Announce)
```

Fig. 1. Implementing (∞, ∞, t)-cons from $(\infty, t+1, t)$-cons and test&sets.

Proof. The simple construction in Figure 2 implements a (∞, ∞, ∞)-test&set object from the infinite array $B[1..\infty]$ of $(\infty, 2, 1)$-cons objects and a single doorway bit.

The implementation is quite simple: it treats the $(\infty, 2, 1)$-cons object instances $B[1..\infty]$ as an unbalanced infinite binary tree, where the left child of tree node $B[i]$ is the process that invokes $\texttt{test}_i$, and the right child of $B[i]$ is the contending process (if any) that wins at node $B[i+1]$. Contenders entering as the left child propose the value 0, those from the right propose 1, and each "wins" the node if their proposed value is returned. A familiar 'doorway' bit ensures that the invocation of the eventual test&set winner is concurrent with or precedes the invocation of any test&set loser.

In this construction, an invocation of *invoke*(`test`,i,E) requires at most i operations on the embedded consensus objects–by balancing the tree, as in the adaptive tournament tree of Attiya and Bortnikov [AB00], previously adapted to infinite arrivals by Aspnes, Shah, and Shah [ASS02], this time complexity can be easily reduced to $O(log(i))$. These time bounds have a nice consequence: if processes first invoke a renaming algorithm adaptive to process contention, and use the resulting name in the test&set algorithm (invoking the `test` operation indexed by the new name), the entire construction will be adaptive to process contention. Indeed, there is a one-shot linearizable, wait-free renaming object, (∞, ∞, ∞)-rename, adaptive to process contention, for infinitely many processes using registers [GMT01]. This renaming object supports the operation `rename`, which invoked by process k returns a positive integer i as a new name, where i is linear in the number of invocations to the object. (And distinct invocations return distinct names.) □

```
test_i, returns boolean                                  /* Code of invocation i. */
Shared:
    doorway: boolean, initially 0.
    B[1..∞]: array of (∞, 2, 1)-cons.
Local:
    step: index to B
    result: boolean
1   if doorway then return(0) else doorway := 1 fi
2   step := i
3   result := ¬(invoke(propose,0,B[step]))
4   while ((step ≠ 1) and (result = 1)) do
5       step := step − 1                              /* Step up in tree B. */
6       result := invoke(propose,1,B[step])
    od
7   return(result)                    /* True iff won path to root in B. */
```

Fig. 2. Implementing (∞, ∞, ∞)-test&set from $(\infty, 2, 1)$-cons.

Theorem 3. *There is an implementation of* (∞, ∞, t)*-cons, adaptive to process contention, using registers, one* $(\infty, t+1, t)$*-cons object, and infinitely many* $(\infty, 2, 1)$*-cons objects.*

Since $(\infty, t+1, t)$-cons trivially implements $(\infty, 2, 1)$-cons, Theorem 2 follows.

2.2 Constructing softwired consensus from hardwired

Next we show that it is possible to replace softwired consensus objects with hardwired consensus objects:

Theorem 4. $\forall t \geq 1 : [(t+1, t+1, t)\text{-}cons \Leftrightarrow (\infty, t+1, t)\text{-}cons]$.

This theorem extends known results for the finite case [CHJT94,BGA94]: $\forall n > t \geq 1 : [(t+1, t+1, t)$-cons $\Leftrightarrow (n, t+1, t)$-cons$]$. (These constructions for the finite case do not extend to the infinite. As we remark below, our constructions are adaptive and more efficient when applied to the finite case.)

Theorem 4 follows from:

Theorem 5. *There is an implementation of* $(\infty, t+1, t)$*-cons, adaptive to process contention, using registers and for every set of processes* T*, where* $|T| \leq t+1$*, one* $(|T|, |T|, |T|-1)$*-cons object,* $C[T]$*, such that* $ACL_{pow}(C[T].propose()) = T$*.*

Proof. The construction uses the fact that there is a long-lived, linearizable, wait-free snapshot object, (∞, ∞, ∞)-snap, for infinitely many processes using registers [GMT01], and which is adaptive to operation contention. Such a snapshot object supports two operations: `write`, which invoked by process i updates a variable v_i, and a `scan` which returns the (finite) set of all pairs (i, v_i) such that a `write` to v_i is linearized before the `scan`.

Since we are implementing a $(\infty, t+1, t)$-cons object, out of infinitely many processes at most $t+1$ may eventually participate. (Its access complexity is $t+1$.) However, the identity of the participating processes are not known in advance.

```
propose_i(u: boolean), returns boolean                    /* Code of invocation i. */
Shared:
    C[T]: for every set T of at most t + 1 processes, (|T|, |T|, |T − 1|)-cons.
    S: snapshot object with fields
        level: integer, initially 0,
        value: {⊥, 0, 1}, initially ⊥.
    Result: {⊥, 0, 1}, initially ⊥.
Local:
    snap, oldsnap: finite sets of (process, (level, value)) tuples, initially empty
    toggle: boolean, iniitally 0,
    result: boolean
0   if (Result ≠ ⊥) then return(Result)
1   invoke(write, i, (0, ⊥), S)
2   snap := invoke(scan_i,S)
3   while (participants(snap) ≠ participants(oldsnap)) do
4       result := invoke(propose_i,max(snap, u),C[participants(snap)])
5       invoke(write,i,(|participants(snap)|, result),S)
6       oldsnap := snap
7       snap := invoke(scan_i,S)
    od
8   Result := result
9   return(result)
```

Fig. 3. Implementing $(\infty, t+1, t)$-cons from $(t+1, t+1, t)$-cons.

The algorithm in Figure 3 uses a snapshot object S in which each variable v_i has two fields, $v_i.level$, a natural number, and $v_i.value \in \{\perp, 0, 1\}$. In the `write` operation by process i, denoted by $invoke(\texttt{write}, i, (a, b), S)$, a is written to $v_i.level$ and b to $v_i.value$.

If a `scan` operation returns a set s, define and $val \in \{0, 1\}$, then define:

- $participants(s)$ to be the set of all indices i such that (i, v_i) is in s.
- $max(s, val)$ to be val if there is no tuple (i, v) in s such that $v.value \neq \perp$, and otherwise to be the value of a $v.value$ such that $v.level \geq u.level$ for all (j, u) in s with $u.value \neq \perp$.

Since at most $t+1$ processes participate, eventually the while loop will terminate. Also, because all snapshots of the same size contain the same set of participants, the agreement property of the consensus objects $C[S]$ guarantee that $u.level = w.level$ implies $u.value = w.value$.

Take any complete run α (in which all correct participating processes terminate), and let k be the minimum such that some invocation, $\texttt{cons}_i$, exits the while loop after taking two successive snapshots of size k. We claim that all terminating invocations return the last value r written by $\texttt{cons}_i$ to $v_i.value$. This claim follows from a stronger claim: that in α if any process invokes $\texttt{propose}(u)$ on an object $C[S]$ with $|S| > k$, then $u = r$, and that in any state of α, if $u.level \geq k$ then $u.value$ is either $\perp$ or r. The proof of this claim is by induction on the size of sets S for which processes invoke `propose` operations on $C[S]$,

from k to $t+1$. By the argument above, if $u.level = k$ then $u.value = r$, and the basis follows.

Now suppose the claim holds for all values from k to $k'-1$. If no process invokes `propose` on an object $C[S]$ with $|S| = k'$, then no $u.level$ is ever k' and the induction follows.

Suppose to the contrary that a non-empty set of processes invoke `propose` on an object $C[S]$ with $|S| = k'$, and hence also the result of these invocations may be written to some $u.value$. (By the argument above, all such operations return the same value.)

Some of these processes may propose the non-$\perp$ value from some w such that $w.level = k'$, hence taking as input to $C[S]$ the result from a previous `propose` operation on $C[S]$. But a non-empty subset of processes see only $w.level$ values smaller than k'. Since these processes see a set of participants of size $k' > k$, they also see that $v_i.k$ is set to k. Hence, they will propose to the consensus object $C[S]$ a non-$\perp$ value of some $w.value$ with $w.level$ at least k but less than k', which by induction is r.

Note that any correct (hence termination) invocation either sees a value set in the *Result* register, or enters and exits the while loop on line 3. The *Result* register is only set by invocations that exit the same while loop, so it suffices to consider the values of *result* in the latter invocations when they exit the while loop. By definition, all such invocations see at least k participants, and by the claim above, $result = r$.

Finally, note that the number of invocations to the embedded snapshot object by any process is linear in the process contention. Since the snapshot is adaptive to operation contention, the entire construction is adaptive to process contention. □

As noted at the beginning of this section, using this construction from hardwired objects, the $(\infty, t+1, t)$-cons object in Figure 1 can be replaced with hardwired objects. Moreover, the softwired $(\infty, 2, 1)$-cons objects embedded in the test&set objects can also be replaced with hardwired objects, proving the next result. (The simultaneous access complexity of the construction is $t+1$ and hence it matches the lower bound of Section 4.)

Theorem 6. *There is an implementation of (∞, ∞, t)-cons, adaptive to process contention, using registers and one $(|T|, |T|, |T|-1)$-cons for every set of processes T where $|T| \leq t+1$, and infinitely many $(2, 2, 1)$-cons objects.*

Designed for an infinite number of processes, this construction is more efficient than previous constructions when the number of processes is finite. Prior constructions have complexity exponential in n and t [CHJT94,BGA94]:

Corollary 1. *There is an implementation of (n, n, t)-cons, adaptive to process contention, using registers, one $(|T|, |T|, |T|-1)$-cons for every set of processes T where $|T| \leq t+1$, and $O(tn^3)$ $(2, 2, 1)$-cons objects.*

(We note that a non-adaptive version of this construction, without renaming, uses only $O(tn^2)$ $(2, 2, 1)$-cons objects.)

If $n > t$, (n, n, t)-cons objects trivially implement $(t+1, t+1, t)$-cons. Hence, Theorem 6 and Corollary 1 establish the equivalence of t-resilient and wait-free consensus for both infinite and finite numbers of processes.

These are strong equivalences compared to the similar result for finite numbers of processes alluded to by Chandra *et al* [CHJT94], which requires the base objects in the wait-free construction to be wait-free and soft-wired, so that the n processes can "simulate" them. Our constructions run these objects as black boxes, and need no such assumptions.

3 The number of faults is not known or may be infinite

A major open question is the relationship between t-resilient consensus and wait-freedom for infinitely many processes: The result $[\forall t \geq 0 : (t+1, t+1, t)\text{-cons}] \Leftarrow (\infty, \infty, \infty)$-cons is trivial, but what of the other direction? The major result of this section shows the converse, but only in runs with bounded concurrency:

Theorem 7. $[\forall t \geq 1 : (t+1, t+1, t)$*-cons*$] \Leftrightarrow (\infty, \infty, \infty)$*-cons in runs with bounded concurrency.*

This theorem is a corollary of the following result:

Theorem 8. *There are implementations of* (∞, ∞, ∞)*-cons for bounded concurrency, using registers and either:*

1. *for every resilience bound* t, $t \geq 1$, *one* $(\infty, t+1, t)$*-cons object, or*
2. *for* every *finite set of processes* T, *one* $(|T|, |T|, |T|-1)$*-cons object.*

Proof. Focusing on the second part of the theorem, as in Theorem 5, we use the construction from Figure 3, but adding as shared objects hardwired $(|T|, |T-1|)$-cons objects $C[T]$ for *every* finite set T (not just for those with $|T|$ bounded by $t+1$).

Termination of the while loop is assured by the bounded concurrency assumption: the *Result* register blocks more than a finite number of invocations from entering the while loop. Otherwise the proof is identical. (The *Result* register and lines 0 and 8 were not necessary for the previous case, where the number of invocations was bounded by a known t.)

The first part of the theorem simply substitutes a single $(\infty, t+1, t)$-cons object in place of the (infinitely many) hardwired objects for sets of size $t+1$. □

As noted, it is an interesting open question whether the bounded concurrency assumption is necessary in Theorem 7, or can it be replaced with unbounded concurrency. An interesting weaker question is also open: whether consensus for unbounded concurrency can be implemented from the set $\{(t+1, t+1, t)$-cons:$t \geq 1\}$, for a finite but unknown number of faults.

4 Lower bounds: The Simultaneous-Access Theorem

We first state a general theorem establishing a necessary condition for implementing consensus in shared memory systems. (The proof is a detailed case analysis along the lines of previous proofs, cf [FLP85,LA87]. Space constraints preclude inclusion of details.) We show that when at most t processes may crash, the consensus problem is only solvable in systems containing "enough" shared objects on which "enough" processes can simultaneously invoke powerful operations. The theorem shows that there is a tradeoff between simultaneous-access and space complexity: when more processes are allowed to access the same object simultaneously, fewer objects may suffice to implement consensus.

We use the notation *ℓ-participation* to mean that at least ℓ processes must participate. The two extreme cases are: (1) participation is not required (i.e., 1-participation), and (2) participation is required. Participation not required is usually assumed when solving resource allocation problems or when requiring a high degree of concurrency, and is most natural for systems with infinitely many potential participants. (It also has simpler compositional properties in t-resilient constructions than does the participation required model, in which for example, embedded objects must be shown to be accessed by sufficiently many processes to assure invocation termination.)

For any object x, we say x is t-resilient assuming ℓ-participation, if any operation invocation when executed by a correct process, eventually completes in any run in which each of at least ℓ processes participates and in which at most t processes fail.

Theorem 9 (The Simultaneous Access Theorem). *In any implementation of a t-resilient consensus object for any number (finite or infinite) of processes, assuming ℓ-participation and $t \geq 1$, for **every** set of processes L where $|L| = max(\ell, t+1)$ there is **some** set $T \subseteq L$ where $|T| = t+1$, such that for some object o, all the processes in T can **simultaneously** invoke powerful operations on o.*

We explicitly state two interesting special cases:

Theorem 10. *Any implementation of a (∞, ∞, t)-cons where $t \geq 1$, must use,*

- *when participation is required, for **some** set of processes T where $|T| = t+1$, at least one object on which the $t+1$ processes in T can simultaneously invoke powerful operations, and*
- *when participation is not required, for **every** set of processes T where $|T| = t+1$, at least one object on which the $t+1$ processes in T can simultaneously invoke powerful operations.*

The last requirement may be satisfied by having just one object, whose access control list includes all the processes, and on which every subset of $t+1$ processes can *simultaneously* invoke powerful operations. The following observations follow from the last theorem.

Corollary 2. *Any implementation of (∞, ∞, t)-cons where $t \geq 1$, from registers and t'-resilient consensus objects where $t' \leq t$, requires at least:*

- *When participation is required, one $(u, t+1, t)$-cons for some $u > t$.*
- *When participation is not required, for every set of processes T where $|T| = t+1$, a $(u, t+1, t)$-cons for some $u > t$, which all the processes in T can access.*

Corollary 3. *Any implementation of a wait-free consensus object for infinitely many processes from registers and wait-free consensus objects for finitely many processes requires, for every positive integer number k, $(u, t+1, t)$-cons for some $u > t \geq k$.*

5 Universal constructions for infinitely many processes

Earlier work on fault-tolerant distributed computing provide techniques (called *universal constructions*) to transform sequential specifications of arbitrary shared objects into wait-free concurrent implementations that use universal objects [Her91,Plo89,JT92]. Plotkin showed that *sticky bits* are universal [Plo89], and independently, Herlihy proved the universality of consensus objects [Her91]. Herlihy also classified shared objects by their consensus number: that is, the maximum number of processes that can reach consensus using multiple instances of the object and read/write registers [Her91]. In their work on randomized consensus for infinitely many processes, Aspnes, Shah, and Shah mention simple modifications to Herlihy's universal construction for crash faults [Her91] to accommodate the case that that the number of participating processes may be infinite [ASS02]. As in [MMRT03], it is also possible to generalize the definition of t-resilient object for which this construction is valid. (Herlihy's universal construction is wait-free for fixed n and implements any object with a sequential specification. In a t-resilient setting, objects with inherently concurrent behaviors may be extremely useful. For example, once $t+1$ active processes have been identified, algorithms can safely wait for one of these (necessarily correct) processes to announce the result of a computation–as in the algorithm in Figure 1.) Due to space constraints, we omit the full details of this generalization, and assuming some familiarity with Herlihy's construction, outline our interpretation of the changes alluded to by Aspnes et al [ASS02], before stating their consequences for our setting.

The key idea of Herlihy's universal construction for n processes is for each process to first announce its next invocation to a single-writer shared register, then to compete (using consensus) to *thread* (a binary encoding of) its id to a sequence of such ids. The state of the object can be inferred by reading this sequence of threaded process ids, then mapping those via the shared register to a sequence of invocations. To ensure every process invocation is eventually threaded, each process *helps* others by threading another process before terminating.

Specifically, to implement the jth invocation by process i, a description of the invocation is first written in the shared register $\mathsf{Announce}[i][j]$. Process i contends with the other processes to thread this invocation (and that of another invocation by process k during the helping stage) by adding a binary encoding of i (correspondingly, k) to $\mathsf{Sequence}[1...]$, of process-id's, where each $\mathsf{Sequence}[k]$ is a $\lceil \log(n) \rceil$ string of $(n, n-1)$-cons objects.

As suggested by Aspnes, Shah, and Shah [ASS02], the first modification necessary is to specify $\mathsf{Sequence}[k]$ as a consensus object over an infinite set of values. Using an unbalanced, infinite binary tree, with binary consensus objects as internal nodes, the values tracing a path from leaf to root encodes the corresponding input, extending binary consensus to an infinitary domain.

The second modification necessary is to carefully specify the order in which processes help other invocations–and to ensure that each invocations looks to help a pending invocation (if one exists) earlier in that order, before it seeks to thread its own id. The simplest choice is for processes to invoke adaptive renaming for each invocation, so that the j'th invocation by process i is mapped to a new unique name k, bounded by a function of the previous and concurrent invocations. The values of i and j can be recorded in the k'th entry of a new array, $Name\_to\_invocation[k]$. This way, each invocation has a finite number of preceding invocations in the order. Once a process i announces it's j'th invocation, the (finite) number of pending invocations may be threaded ahead of it, together with the finite number of invocations that precede it in this order (only of course if they are invoked). After this, any other invocation will choose to help this one.

Corollary 4. *We have the following, assuming infinitely many processes and* $t \geq 1$*:*

1. $(\infty, t+1, t)$*-cons objects are universal for* t*-resilient objects.*
2. $(t+1, t+1, t)$*-cons objects are universal for* t*-resilient objects.*
3. *The infinite set of objects* $\{(\infty, t+1, t)\text{-cons}\}$ *is universal for wait-free, bounded concurrency objects.*
4. *The infinite set of objects* $\{(t+1, t+1, t)\text{-cons}\}$ *is universal for wait-free, bounded concurrency objects.*

Proof. The first part of the corollary follows by implementing the consensus objects in $\mathsf{Sequence}$ using Corollary 3, the second using Theorem 5. The last two parts follow from Theorem 8.

We note also that the referenced corollary and theorems, together with the use of renaming, support implementations that are adaptive to operation contention. □

References

[ASS02] J. Aspnes, G. Shah, and J. Shah, Wait-free consensus with infinite arrivals. In *Proc. 34th Annual Symp. on Theory of Computing*, 524–533, May 2002.

[AB00] H. Attiya and V. Bortnikov, Adaptive and efficient mutual exclusion. In *Proc. 19th ACM Symp. on Principles of Distributed Computing*, 91–100, July 2000.

[BGA94] E. Borowsky, E. Gafni, and Y. Afek. Consensus power makes (some) sense! In *Proc. 13th ACM Symp. on Principles of Distributed Computing*, 363–372, August 1994.

[CHJT94] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg, Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies. In *Proc. 13th ACM Symp. on Principles of Distributed Computing*, 334–343, August 1994. Expanded version:
`www.cs.toronto.edu/ vassos/research/list-of-publications.html`.

[CM02] G. Chocker and D. Malkhi. Active disk paxos with infinitely many processes. In *Proc. 21th ACM Symp. on Principles of Distributed Computing*, 78–87, July 2002.

[FHS98] F. Fich, M. Herlihy, and N. Shavit, On the Space Complexity of Randomized Synchronization. *Journal of the ACM*, 45(5):843–862, September 1998.

[FLP85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GMT01] E. Gafni, M. Merritt, and G. Taubenfeld. The concurrency hierarchy, and algorithms for unbounded concurrency. In *Proc. 20th ACM Symp. on Principles of Distributed Computing*, 161–169, August 2001.

[HW90] M.P. Herlihy and J.M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems* 12(3):463–492, July 1990.

[Her91] M.P. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems* 13(1):124–149, January 1991.

[JT92] P. Jayanti and S. Toueg, Some results on the impossibility, universality and decidability of consensus, In *Proc. 6th Int. Workshop on Distributed Algorithms, (WDAG'92)* LNCS 647, 69–84. Springer Verlag, November 1992.

[LH00] W.K. Lo and V. Hadzilacos. On the power of shared objects to implement one-resilient consensus. *Distributed Computing* 13(4):219–238, 2000.

[LA87] M.C. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[MMRT03] D. Malkhi, M. Merritt, M. Reiter, and G. Taubenfeld. Objects shared by Byzantine processes. *Distributed Computing* 16(1):37–48, 2003. Also in: *Proc.14th International Symposium on Distributed Computing (DISC 2000)*, LNCS 1914, 345–359, 2000.

[MT93] M. Merritt and G. Taubenfeld. Speeding Lamport's fast mutual exclusion algorithm. *Information Processing Letters*, 45:137–142, 1993. (Also published as an AT&T technical memorandum, May 1991.)

[MT00] M. Merritt and G. Taubenfeld. Computing with infinitely many processes. *Proceedings of the 14th International Symposium on Distributed Computing* LNCS 1914, 164–178. Springer Verlag, October 2000.

[Plo89] S.A. Plotkin. Sticky bits and universality of consensus. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, 159–175, August 1989.