Voting and Collation in Distributed Systems Middleware

Dave Bakken[1]

School of Electrical Engineering and Computer Science

Washington State University

Pullman, Washington USA

www.eecs.wsu.edu

Seminar at Georgia Tech.

February 20, 2001

WASHINGTON STATE UNIVERSITY

[1] In collaboration with Zhiyuan "Troy" Zhan, Chris Jones & David Karr (BBN), and Doug Blough (Georgia Tech)

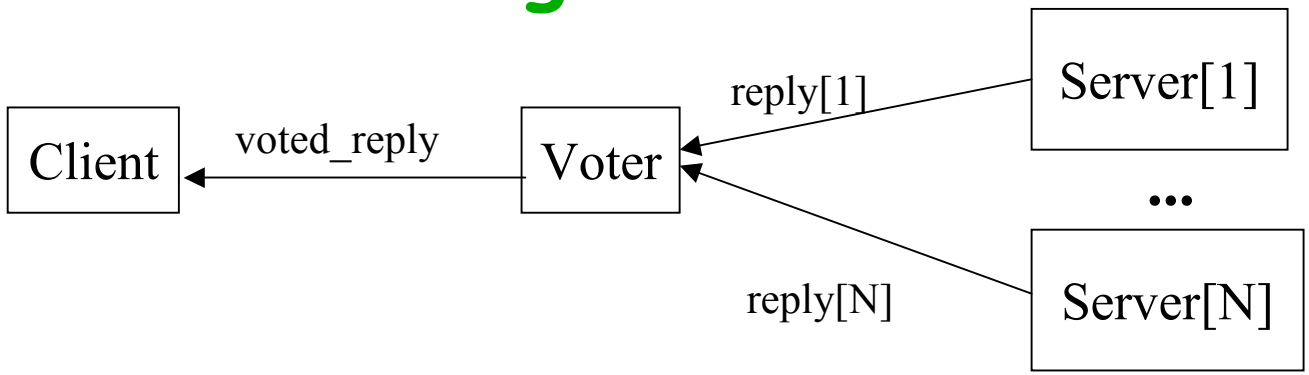
Outline of Presentation

- **Overview of voting and collation**
- Voting Virtual Machine (VVM) architecture
- Voting Algorithm Analysis and Voter Management
- Voting Status Service
- Related work
- Conclusions

Motivation

- The increasing demand of availability of online services can be met with replication of servers.
- One of the most common replication strategies is called active replication.
- In active replication, a request it sent to all of the replicas and each replica will process the request
- Each replica will then send their replies back to client and one reply must be chosen for the client to use.
- This selection process is called voting
- Collation is more general than voting
  - No replicated servers
  - Values not necessarily supposed to be identical
  - Seems to be a fundamental problem in distributed systems

Voting Overview



Voting Definitions:

- collation voting: choosing one reply (or request) from many
  - Differs from synchronization voting, a form of pessimistic concurrency control, also called "voting" in the literature
- ballot: one request or reply from one object replica (reply[1])
- vote: the process of choosing one ballot from many
- byte-by-byte voting: voting algorithms compare marshaled parameter buffers on a byte-by-byte basis, unaware of data types, alignment, etc.
  - More on this later…

Example

```
interface foo {
    long method1 (in long a);
    void method2 (in long d, inout short e, out double f)
}
```

Method	Direction	Voting "params"	
Method1	Request	{a}	Can't do today!
Method1	Reply	{**rtn**}	
Method2	Request	{d,e}	Can't do today!
Method2	Reply	{e,f}	Can't do today!

- **No voting on replies or request parameters today**
- **Byte-by-byte voting is very fragile and cannot be correctly implemented (except rtn for int)!**
  - **More on this later...**

What is Needed

- Middleware support for voting
  - Allow voting on any parameter
  - Separate voting language for reuse, analyzability
  - Hooks to allow changing of voting algorithms at runtime
  - Voting management to provide adaptivity and client transparency
- Such middleware applies to more than just voting: collation & data fusion
  - Intrusion detection: multiple different probes monitoring a host/LAN/domain
  - Ad hoc mobile network protocols
  - Distributed sensor networks
  - Hierarchical resource monitoring (Astrolabe from Cornell)
  - Merging object state from cached copies or partitioned replicas
- Research issues for non-voting collation/fusion
  - Naming
  - When is a vote over? How many ballots are expected?
  - Creating a family of middleware and voting language for different form factors, including even maybe with VHDL/FPGA

Outline of Presentation

- Overview of voting and collation
- **Voting Virtual Machine (VVM) architecture**
  - **Voting Virtual Machine**
  - **Voting Description Language**
- Voting Algorithm Analysis and Voter Management
- Voting Status Service
- Related work
- Conclusions

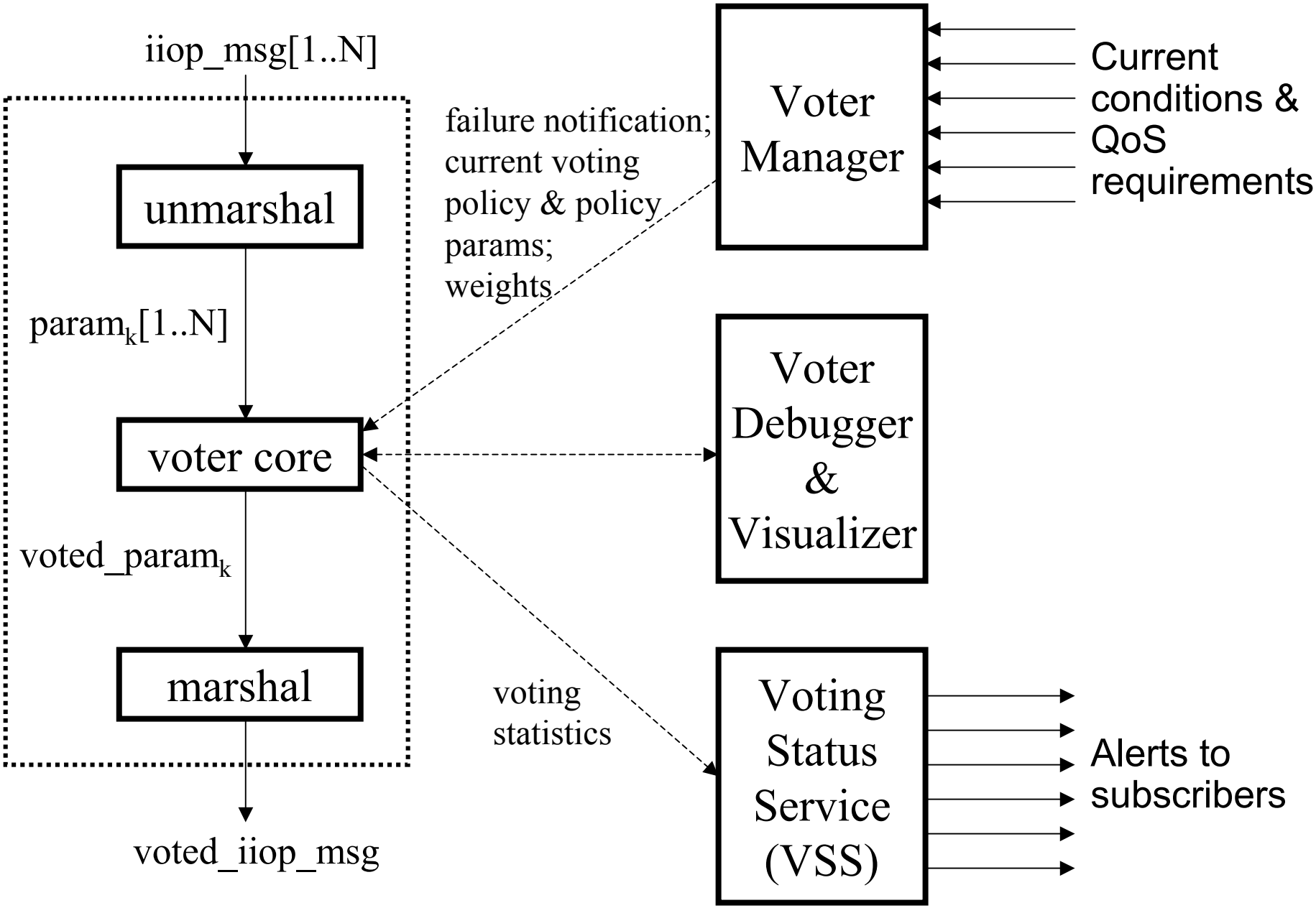
Voting Virtual Machine (VVM)

Novel features:

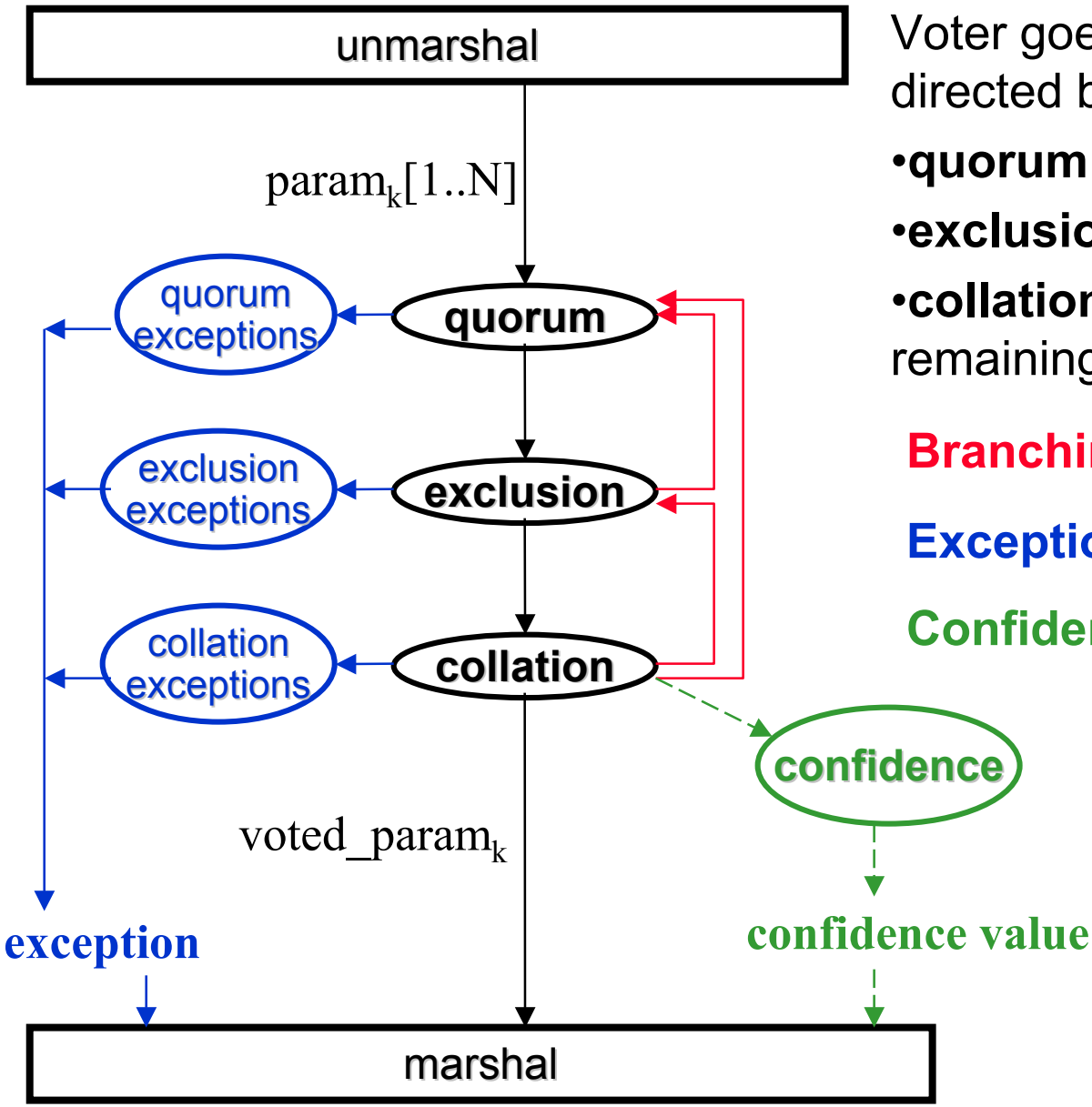
- VVM performs voting on application parameters in a unmarshaled network message (not on a marshaled parameter buffer)
  - Nobody has done anything like this
  - Embeddable into CORBA, DCOM, other kinds of middleware (MOM, publish-and-subscribe, XML-RPC, ...)
  - Voting module thus <u>not</u> embedded in the application
- Voting Description Language (VDL) allows the coding of portable, reusable, stand-alone voting algorithms
  - Nobody has done anything like this
  - Supports both *static voting* and *dynamic voting* (accounts for membership changes and group size)
  - Supports spectrum of tradeoffs between
    - Performance
    - Fault tolerance
    - Precision/correctness

    Note: Application programmers and users care about these QoS properties, not the details of a voter algorithm!

VVM Architecture



VVM Voter Core States & VDL Primitives



Voter goes through 3 states, as directed by voting policy (in VDL):

- **quorum**: wait to vote
- **exclusion**: toss out some ballots
- **collation**: choose one of the remaining

Branching

Exceptions

Confidence Values

Simple VDL Examples

- Example # 1:
  - English: “wait until 4 ballots have arrived, exclude the lowest one, then choose a random one from those left”
  - VDL: **quorum** (**until (**4)) **exclusion (lowest (**1)) **collation (random)**
- Example #2:
  - English: “wait until half of the ballots have arrived, exclude a random one, then choose the median of those left”
  - VDL: **quorum** (**until** (50 %)) **exclusion (random (**1)) **collation (median)**
- Example #3:
  - “wait until all but 2 of the messages have arrived, exclude the two highest-valued ones, then choose the most common one left”
  - VDL: **quorum** (**all_but (**2)) **exclusion (highest (**2)) **collation (mode)**
- **VDL above is simple, but is much less simple with**
  - **Exceptions**
  - **Branching**

  And will be less simple with
  - **Multiple-parameter collation**
  - **Patterns useful for multiple method signatures**

  but will hopefully stay as readable!

VDL Primitives

- quorum
  - **until** *k* [%]
  - **all_but** *k*
  - **random** *n m* [%]
- exclusion
  - **lowest** *n* [%]
  - **highest** *n* [%]
  - **furthest** *n* [%]
  - **distance** *e* [%]
  - **outside_sigma** *x*
  - **distance_neighbor** *d*
  - **distance_cluster** *d*
  - **cluster_support** *d p [c]*
  - **inner** *k*
  - **nearest** *k*
  - **random** *n* [%]
  - **none**
- collation
  - **median**
  - **mean**
  - **mean_neighbor**
  - **mode**
  - **random**
- Exceptions: conditions based on
  - Elapsed time since first ballot
  - Number excluded
  - Percent excluded
  - Number remaining after exclusion
  - Standard deviation after exclusion
  - …
- Confidence values
  - Above conditions for exceptions
  - More TBD …work in progress….

VDL Syntax Overview

```
policy name [ (parameters) ] {

    quorum (quorum_op (params))
        [throw ex_name if (condition) ]*

    [exclusion  [label] (exclusion_op (params))   [replace by mean|median|(value)]
        [ [throw ex_name if (condition) ]
          [goto quorum [(using (quorum_op (params)) more|total)] if (condition) ]  ]*
    ]+

    collation    (collation_op (params))
        [ [throw ex_name when (condition) ]
          [goto quorum [(using (quorum_op (params)) more|total)] if (condition) ]
          [goto exclusion [label] [(using (exception_op (params)))] if (condition) ]   ]*

    [confidence  (confidence_expression) ]
}
```

SuperMajority Example

// Return a ballot if 60% of the ballots are within 1% of the median of the ballots
// received so far, but only wait 5 seconds. Ballot returned should be median
// from those within 1%

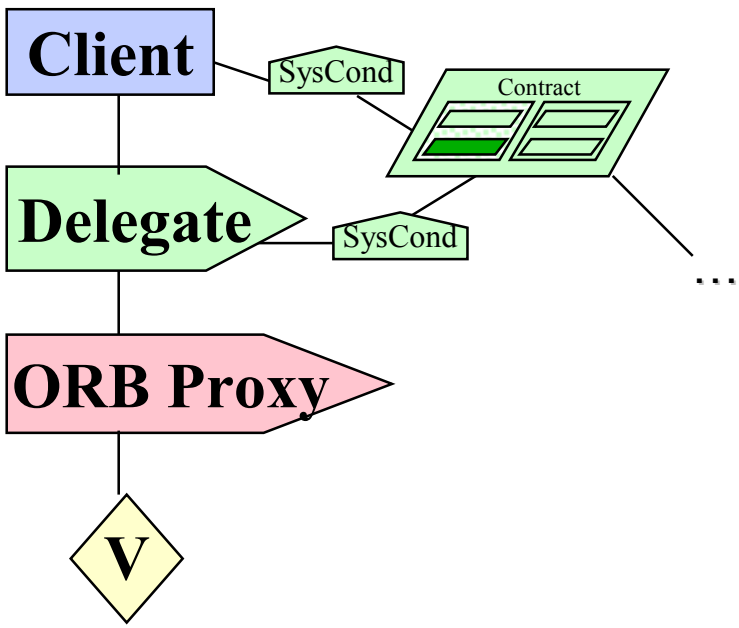
```
policy SuperMajority {

        quorum (until (60 %))
            throw QUORUM_TIMEOUT if (time_since_ballot(1) > 5000)

        exclusion (distance  (1 %))
            throw BAD_VALUES if (pct_excluded_total > 40)
            goto quorum (using (until (60%)) total)  if (pct_remaining_total < 60)

        collation (median)

        confidence (ballots_remaining / ballots_received)
}
```

Voting on Non-Basic Types

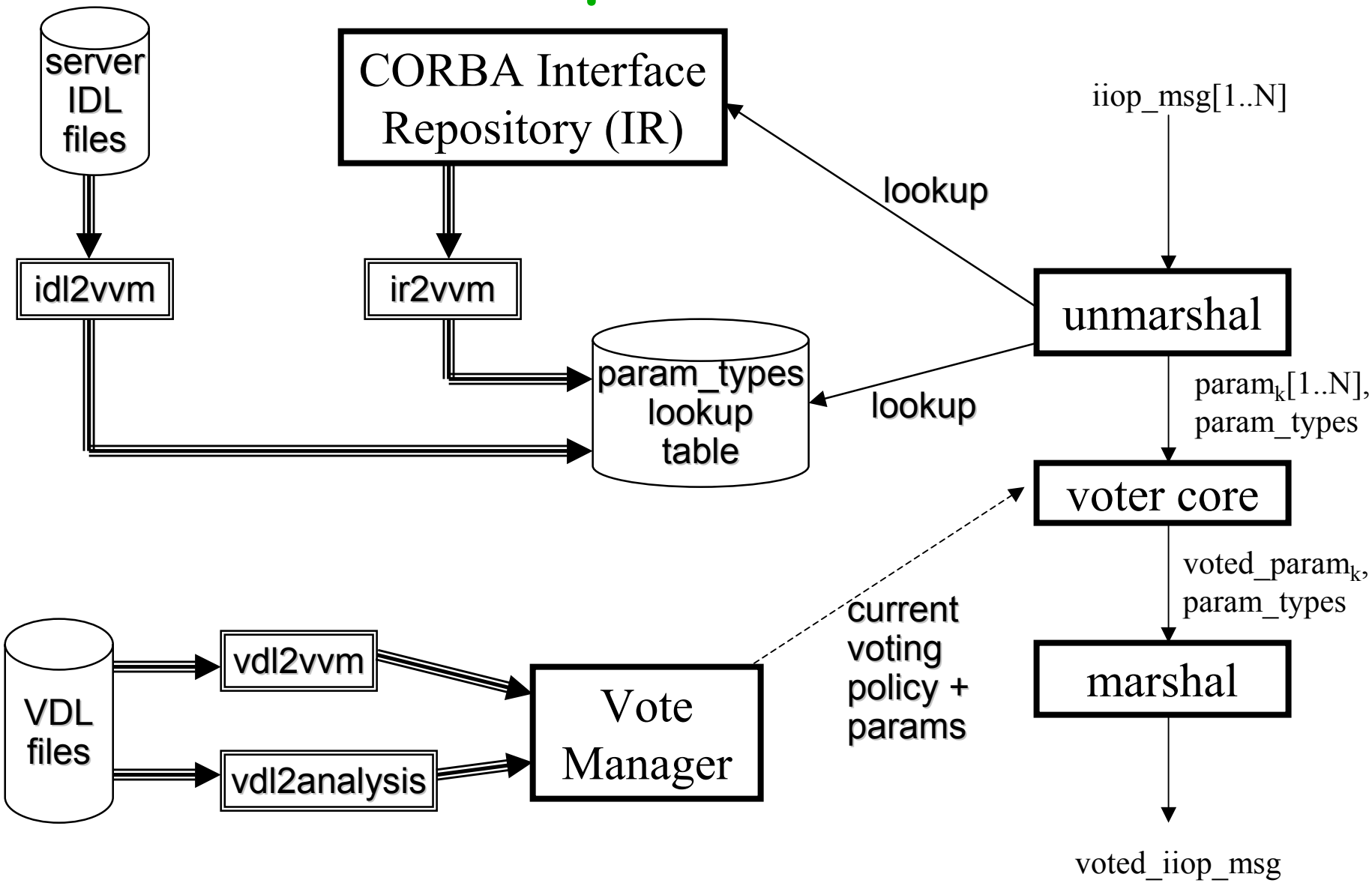
- Problem: how to vote on parameters of types other than basic types (**long**, **float**, …)
  - VVM cannot know about them without being told!
  - We assume type is defined as a CORBA IDL struct
- Solution: allow definers of the types to tell us how to vote with helper objects
- Three different APIs for helper object to implement (in Java), with greater info provided for greater control
  1. Maps to and from struct to **double**
  2. Implements Java *comparable* interface, plus operations which do not make sense by comparing alone (**mean**, **mean_neighbor**)
  3. Implements methods for all **exclusion** and **collation** operations
- Also need to provide a marshal object for the struct
- Need to fill in a configuration table with struct and class names for above
- This voting on structs can be used for object state (caching, merging partitioned replicas)

Confidence Values

- Problem: returning a vote or an exception are two extreme choices!
- Issue: how “good” was the vote?
- Confidence values used in neural networks and fuzzy logic
- Idea for VVM:
  - Return another value(s) “on the side”, like Unix errno
  - Optionally used by client or perhaps a QuO delegate or other intermediate layer to adapt…



VVM Development Environment



VVM Failure Model

- Client and server hosts: crash failures
- Server applications: Byzantine
- Server systems software: crash
  - Could secure with network attachment controller *a la* Delta-4 NAC
  - Could use stronger Byzantine protocols *a la* Rampart

Outline of Presentation

- Overview of voting and collation
- Voting Virtual Machine (VVM) architecture
- **Voting Algorithm Analysis and Voter Management**
- Voting Status Service
- Related work
- Conclusions

Voting Algorithms and Analysis

- Goal: understand all/most known voting algorithms, to see which are:
  1. Expressible in VDL as it now is
  2. Expressible in VDL with some VVM extensions (e.g., new operation)
  3. Needs architectural/plumbing changes
  4. Not expressible in VVM in any reasonable use of it

  Initial results: many algorithms ("most" of literature?) seem to fall into #1 or #2
- Analysis (work just beginning)
  - Observation: in essence, a voting algorithm gives a probabilistic tradeoff between correctness and performance and fault tolerance, for a given set of conditions:
    - Latency and bandwidth
    - Number and kind of failures
    - Etc.
  - Goal: develop analysis tools that can ascertain this tradeoff for a VDL policy, over a range of conditions
    - May have to do over a subset of VDL, but which subset, and why?
  - Then use this analysis to allow manager to choose the best (or at least a reasonable) algorithm for the current conditions

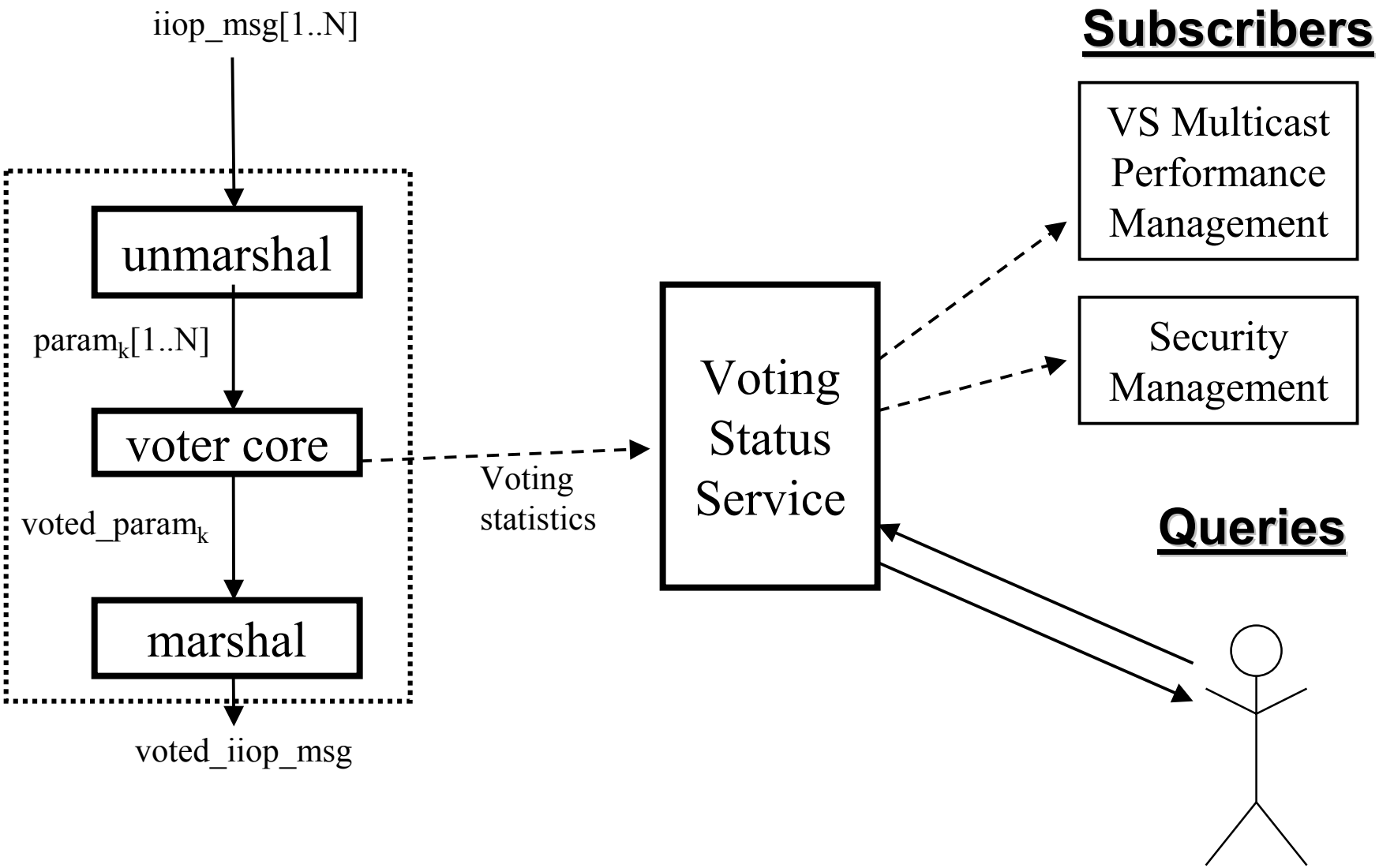
Voter Management

- A voting manager chooses a voting policy and change it when needed
  - Also can decide the weights for each server/replica
  - Tries to meet current QoS requirements
  - Provides voting transparency
  - Provides adaptive voting
- Hierarchy of managers with higher managers giving lower managers a more global view of system and requirements
- Voter managers based on input from
  - Network management giving utilization & capacity
  - Security management regarding a failure in system
  - Intrusion detection system as to which hosts or domains may be compromised
  - Higher-level voting managers
  - QoS contract
    - Allows user or QoS manager to tell the current preferred tradeoff between precision/correctness and performance and fault tolerance
- Our voter management work is just beginning…

Outline of Presentation

- Overview of voting and collation
- Voting Virtual Machine (VVM) architecture
- Voting Algorithm Analysis and Voter Management
- **Voting Status Service**
- Related work
- Conclusions

Voting Status Service (VSS)



Voting Status Service (cont.)

- Observation: voter tracks a lot of information which can be useful to others
- VSS is a first-class service offering this
- VSS gets low-level updates from voter core
- VSS maintains moving averages of various conditions; clients of VSS can parameterize this moving average and threshold for their alert
- Supports multiple interaction paradigms:
  - Registering for a (possibly compound) condition, getting a callback when to crosses the given threshold
  - Interactive queries of current conditions
- Example conditions
  - A given replica in a given group is late “too often”
  - Any replica in a given group is late “too often”
  - any replica on a given host is late “too often”
  - Too many replicas on any one host are late “too often”
  - Too many replicas in any one domain are late “too often”

Outline of Presentation

- Overview of voting and collation
- Voting Virtual Machine (VVM) architecture
- Voting Algorithm Analysis and Voter Management
- Voting Status Service
- **Related work**
- **Conclusions**

Limitations of Byte-by-Byte Voting

- CORBA provides interoperability across
  - CPU architecture
  - Operating system
  - Programming language
  - ORB implementation
- CORBA's CDR uses IEEE 754 encoding for floating point values' transmission, but with different architectures (and other 3 variables same) can have different internal precisions and roundoff differences, …
- Have to look at the IDL to be able to handle
  - variable-length header information (e.g., system context) which an ORB implementation may fill in or leave out (else alignment off and undetected padding bytes)
  - CORBA spec states value of padding bytes is undefined, anyway!
- Byte-by-byte voting in practice:
  - Developers have no problems in lab or single LAN, because very homogenous (CPU, OS, language, ORB)
  - When fielded or released, mysterious bugs start to show up because much more heterogeneous
  - This is true for all middleware, not just CORBA!

Related Work

- Synchronization Voting
  - First by Thomas of BBN (1979)
  - Weighted Voting (Gifford, 1979)
  - Generalizations: multidimensional, etc.
  - Mostly (except NMR) for databases
- Collation voting (all byte-by-byte)
  - N-Version programming
    - no votes on replies, parameters
    - very limited quorums
  - NMR systems
  - Recent CORBA replication: AQuA, Orbix+Isis, Eternal, Electra
    - no vote on replies, parameters
    - very limited quorum, etc.
  - Immune (UCSB, 1999)
    - Does allow voting on client requests and reply parameters
    - Says for a majority “being identical in value” (I.e., byte-by-byte)

Harsh Questions: Java vs VDL

- Why not just define a voting API and upload Java?
- Is VDL just “distributed systems syntactic sugar”?
- Is the VVM clever and cute, but just “stupid middleware tricks”?

- VDL has many advantages
  - Analyzability and the promise of VVM manageability
    - Adaptive voting
    - Transparent voting
  - Readabilty of VDL code, so better reuse
  - Higher-level language for coding voting algorithm
  - Termination of VDL code and other safety properties should be able to be guaranteed (work in progress…)
  - ….

Conclusions

- VVM can be embedded in any kind of middleware we are aware of
- VVM performs voting on actual application-level data parameters, not marshaled parameter buffer in a byte-by-byte fashion
- VDL allows specification of voting algorithms which are
  - flexible
  - portable
  - reusable
- Voter management offers the hope of offloading the burden of managing and even the awareness of voting away from the application programmer
- VSS provides status service on different voting conditions, for performance management, security management, etc.

Backup Slides

- Security of Voting Algorithms
- Other Ongoing Research
- Parameterizing VDL policies
- VDL Expressiveness of Space of Voting Algorithms

Random & Weighted Voting in VDL

- Random operations for each of the 3 states of the voter core
  - **quorum**: wait for a random number of ballots (in some range)
  - **exclusion**: randomly exclude some number of ballots
  - **collation**: randomly choose one of the ballots
- Weighted voting allows non-equal treatment of ballots from different replicas
  - Main use: security in voting on (supposedly) identical replicas
  - Also can also add randomness to collation and data fusion
- Operations currently in 2 voter core states:
  - **quorum**: wait for points, not ballots, where each ballot's arrival is >= 1 point
  - **collation**: expand remaining ballots based on a weighting, then do mode, median, ...
  - collation example: if ballots contain {2,3,4} and weights are {1,2,2}, then expand to {2,3,3,4,4} and then do median or mode or mean or random or ...
  - **exclusion**??? Work in progress... Could be similar to collation expansion

Tolerating Faulty Values

- Value attack: an attempt by an adversary to corrupt a vote by injecting ballots with bad values
- Value failure: a ballot with one or more bad parameter values
- Observation: the VVM has the ability to detect faulty application-level data values.
- Questions
  - How good is the VVM at thwarting or impeding value attacks?
  - How many value failures can the different VDL policies tolerate without the vote being bad?
- Bottom line:
  - The value attacks that can be tolerated range widely across VDL space
  - Many VDL algorithms offer good value attack tolerance
  - Weighted voting with intrusion detection offers much promise
  - See VVM thesis (5/2000) for more details

Resilience against Malicious Attacks

- Questions
  - Can adversary learn the voting policy in use?
  - If yes, can it be useful to them in their attempts to corrupt the votes?
- Assumptions
  - Adversary can read ballots and voted answer messages
  - Voting on only one parameter and adversary knows which one
  - Knows the order in which the ballots were received by voter
  - Adversary has not corrupted any of the replicas
  - Adversary can not decode VDL sent by manager to voter core
- Observations
  - Adversary might start looking at ballots and final answer for patterns
  - If no patterns are determined, a brute force approach might be next
- Bottom line: VDL space classified by the following
  - Can always determine the VDL in one vote
  - Can sometimes determine the VDL in one vote
  - Must always wait for two or more ballots to determine VDL

Ongoing Security, Middleware, and Replication Research

- Security and Survivability
  - Continue research on security of voting algorithms
  - Voting Status Service as an Intrusion Detection Feed
  - Using intrusion detection to determine replica weighting, by manager
  - VVM for use in Intrusion Detection (Hummer at U. Idaho)
- Middleware Heterogeneity Issues
  - Heterogeneity across CPU, language, OS, middleware vendor
  - Need to investigate these issues with at least CORBA and SOAP, DCOM's heir apparent
- Replicating the Voting VM
  - We have a simplified passive replication scheme.
  - Looking at active replications which has the ability to tolerate value errors
  - Evaluate the cost and benefits of different replication strategies and use that info in designing the voting managers so that they can dynamically change strategies.
    - Probably passive will be the best, but knowing exactly why is useful

Architectural Diversity

- VVM is good for more than just (expensive) active replication….
- Caching
  - high availability, but staleness or inconsistency
  - state of instance of an object is cached, we will implement this as a parameter type extension and be used to merge multiple copies whose replicas have been lost, to create new base replica
- General Collation Engine
  - use of VVM as collation engine where values are not presumed to be identical
  - Issue #1: when is “vote” over?
  - Issue #2: naming of senders of ballots (not “replicas” any more)
- Quorum Consensus (esp. newer work with byzantine coverage)
- Merging Partitioned Subgroups (VVM as a comparison engine)
- Distributed Sensor Networks
  - Hot research area!
  - Plan to develop families of VVM and VDL for different memory and power constraints

VDL Built In Types for Parameterization

```
enum QuorumOps {until, all_but, random, ..};

enum ExclusionOps {lowest, highest, …, random, none};

enum CollationOps {median, mean, mean_neighbor, mode, random};

enum VDL_param_type {is_double, is_long, is_absent};

// a param is either long or double; and can also have '%'
struct VDL_params {
    double p1_double; long p1_long; VDL_param_type p1_type; boolean p1_pct;
    double p2_double; long p2_long; VDL_param_type p2_type; boolean p2_pct;
    double p3_double; long p3_long; VDL_param_type p3_type; boolean p2_pct;
};
```

Parameterized Supermajority

```
subtype Super_exclusion_ops: ExclusionOps {lowest, furthest};
subtype Super_quorum_increment: long [2,4];  // how many more to wait for

policy Supermajority2 ( double pct_same, double pct_for_confidence,
                        Super_quorum_increment q_inc,
                        Super_exclusion_op ex_op, VDL_params ex_params,
                        CollationOps c_op) {

        quorum (until (pct_same %))
            throw QUORUM_TIMEOUT if (time_since_ballot(1) > 5000)

        exclusion (ex_op (ex_params))
            throw BAD_VALUES if ((pct_excluded_total > (100.0 – pct_same))
            goto quorum (using (until (pct_same %)) total)
                            if (pct_remaining_total < pct_same)
        collation (c_op)
        confidence (ballots_remaining / ballots_received)
}
```

VDL Expressiveness

- Voting Description Language (VDL) allows the expression of all sixteen classes of voting algorithms in most recent voting algorithms survey
  - B. Parhami, “Voting algorithms”, *IEEE Transactions on Reliability*, Dec. 1994.

- Voting algorithm has:
  - N input data objects
  - Each object has weight
  - Producing single output object

	Input	Output
Data	Exact/ Inexact	Consensus/ Compromise
Vote	Preset/ Adaptive	Threshold/ Plurality

VDL Expressiveness (cont.)

- Input Data: what the input object copies are like, compared to each other, assuming no failures
- Exact
  - Values from the replicas are inflexible
    - The value 3.5 does not equal 3.6
  - Default for VDL, no special operations needed
- Inexact
  - Values from the replicas can be considered approximately equal
    - The value 3.5 and 3.6 are close enough to be equal.
  - Supported by VDL with the two exclusion operations, **distance-neighbor** and **distance-cluster**

VDL Expressiveness (cont.)

- Output data: ways to collate from input objects

- Consensus
  - The most common value returned by the replicas
  - VDL supports consensus with the **mode** collation operation

- Compromise
  - Calculating an average answer
  - Support for compromise in VDL is with the use of **mean** and **median**, and **mean-neighbor** collation operations

VDL Expressiveness (cont.)

- Input vote: How flexible the weights are

- Preset vote
  - The weights of the ballots are set at design time
  - Weighting is supported by VVM thus preset is supported

- Adaptive vote
  - The weights of the ballots can be changed at runtime.
  - The VVM supports adaptive votes by allowing the weights of replicas to change at runtime.

VDL Expressiveness

- Output vote: how m any input objects to wait for

- Threshold
  - Vote needs the support of specified number of replicas in order for the voted answer to be considered correct
  - VDL supports threshold voting with collation state exceptions
    - Finer granularity with confidence values
  - Threshold types include: majority, Byzantine, and unanimous

- Plurality
  - Like threshold but relaxes the need for more than half of votes needed.