1 Introduction

Enterprise SY architecture:Cs, Web S, APP Ss, MsgQueue, Pub/Sub SY, DBs Distributed APP developed on sockets. (Fallacies, Interoperability in spite of heterogeneity, Data representation and encoding, Parameter passing convention calling remote procedures, Atomic execution of TX, Enable APP integration, Data persistence) Middleware as layer between transport & application layer or Above OS and below APP. comprises services and abstractions(RMI, Messaging, PubSub,TX,Naming) that facilitate the design, development, and deployment of distributed APP in heterogeneous, networked environments. Deals with interoperability 互通性 , SY integration |C request/response ↔ Middleware ↔ S. |All Computers share single distributed APP and Middleware. Heterogeneity: Computer architecture(big, little endian), OS(Comm. subsystem), Host and network representation of data, Programming language(Representation of characters) Ilities: Reliable; Fault-tolerant; Highly available; Recoverable; Consistent; Scalable; Predictable pref.; Secure; Heterogeneous; Open

8 Fallacies 1.Network Reliable Assumptions:Power Supply, Hard Disk, Node Failures, Config., Bugs Effect:APP hangs, crashes Countermeasures: Redundancy HW&SW systems,middleware &application; Catch Exceptions; Check Codes, React; Retry After Timeouts; pos. neg. ACK; Identify & ignore duplicates; idempotent operations 2.Latency延迟 Is Zero time For Data Transfer(speed Of Light); Bandwidth:how Much Data transferred (bit/s) |Local Call: Push and a Jump to Subroutine; SY Call: OS, 100s of assembly; Call across LAN: SY calls on caller + callee + network latency; Call across WAN: transmission delay 3.Bandwidth is infinite. 4.The network is secure. 5.Topology doesn't change. 6.There is one administrator. 7.Transport cost is zero. 8.The network is homogeneous Standardization: Official:ISO, ITU, DIN, Semi-official: IEEE, W3C, OpenGroup

2 Comm. Basics

ISO OSI: Open Systems Interconnection model, Basis for standards development on SY interconnection, Reference model. |Layers: Application(Peer protocol: HTTP, DNS, FTP); Presentation; Session; Transport: TCP/UDP; Network: IP; Data link; Physical

IPv4 v6(no checksum): Relays datagrams accross networks, Routing enables internetworking, Deliver packet from source to host address, Foundation for TCP/UDP

IP routing:routing protocol, BGP used in internet: Border Gateway Protocol, Routing between AS (Autonomous systems, provider or bigger organization), Exchanges routing and reachability information between AS.

TCP: for HTTP, RPC, Slower than UDP, but with reliability using ACK, Connection oriented protocol, session is initiated, Provides ordering, sequencing, Flow control, sender can't overflow receiver(same speed sending, receiving), PortNum in TCP protocol(not in IP protocol)

UDP: for DNS, Video, Voice, Faster than TCP, Connectionless(no session); Best-effort; Packet independent; No guaranteed delivery; No ordering guarantees; P2P and P2-multipoint.

Ports: a 16 bit number to local host to identify the connection, to differentiate APPs, if packet arrive; Separate for UDP and TCP; $0 - 1023$ reserved to root, $1024 - 65535$ available to regular user; http 80/tcp, ftp 21/tcp, ssh 22/tcp, telnet 23/tcp, finger 79/tcp, snmp 161/udp

APP Layer protocol: Set of rules specifying data transfer between computing end-points (Connection establishment & tear-down Data representation, Comm); DHCP (Dynamic Host Configuration Protocol), HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), Telnet (Telnet Remote Protocol), SSH (Secure Shell Remote Protocol), SIP (Session Initiation Protocol), POP3 (Post Office Protocol 3), SMTP (Simple Mail Transfer Protocol), IMAP (Internet Message Access Protocol), ||browser retreives web page(HTTP 1.1 over TCP, HTTP, FTP, SMTP)

HTTP/1.0 C S: Request: GET <path>/index.html HTTP/1.0 Response: HTTP/1.0 200 OK ErrorResponse: HTTP/1.0 404 Not Found ||HTTP/1.1: HyperText Transfer Protocol, Request/Response protocol for C S comm. on top of TCP (Browser, RESTFUL API's, SOAP over HTTP, NoSQL databases) Methods: GET:cacheable get info by Request-URI HEAD: like GET but MUST NOT return a message-body POST: post a form (bulletin board), uploading data, can be a data-accepting process Responses not cacheable(200 (OK), 204(No Content), 201 (Created), 303 (See Other)) PUT: Enclosed entity to be stored under Request-URI Responses not cacheable DELETE: Delete the resource by Request-URI Response: 200 (OK), 202 (Accepted), 204 (No Content) TRACE: for diagnostics CONNECT: to initialize secure connection, HTTPS on port 443, Proxy is asked to forward the TCP connection |HTTP/1.1 proxy:C GET req. → Proxy, but not in cache. Proxy GET req. → Origin, Origin → Proxy response. C GET req. → Proxy, Proxy → C cached resp. |HTTP/2.0: Better utilization network capacity, Headers compressed, On a single connection req and resp interleaved, Prioritization of requests |||GET/HTTP/1.1 Host:www.tum.de Connection:keep-alive Accept: text/html,application/xhtml+xml,application/xml;q=0.9, image/webp, */*;q=0.8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36 Accept-Encoding: gzip,deflate,sdch |||HTTP/1.1 200 OK Date: Wed, 08 Oct 2014 15:25:53 GMT Server: Apache X-Powered-By: PHP/5.5.17 Content-Encoding: gzip

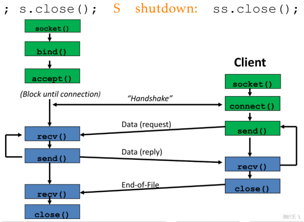
Socket

Move data/Msg/(invoke operation/service and return result/failure) from APP I on Host A to APP K on Host B. Client:Issues req to S(send & receive). Server:Starts up and listens for connections, req.s, and sends/receives. C S eg.: P2P telnet/telnetd, ftp/ftpd (sftp/sftpd), Firefox/Apache.

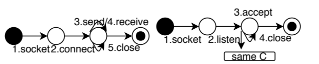
Socket: using network API, network programming abstraction for comm. among processes (APPs) based on (Unix) file descriptors. File/Socket descriptor:int for open file managed by the OS, In Unix any I/O by reading/writing from/to file descriptors. |Socket types: Stream socket:telent, HTTP, browser, java.net.ServerSocket, TCP based(Ordering guaranteed, Error-free); Datagram socket:java.net.DatagramSocket, UDP based; IPv4&IPv6

2 end-points determine a connection: IP address (host address)+Port number

SetUp,UseSockets S Create:
```
final ServerSocket ss = new;/*localhost only*/ss.bind(new InetSocketAddress ("0.0.0.0", port)); while (true) {Socket clientSocket = ss.accept ();/*Block till C*/
```
C connect S:
```
Socket s = new; s.connect(new InetSocketAddress("127.0.0.1", port )); PrintWriter output = new PW(s. getOutputStream()); BufferedReader input = new BR(new InputStreamReader (s.getInputStream())); output.write( req + "\r\n");/*to S*/output.flush();
```
S handle req.: /*in while true*/
```
BufferedReader in = new BR(new InputStreamReader(clientSocket .getInputStream(), Constants. TELNET_ENCODING)); PrintWriter out = new PW(new OutputStreamWriter( clientSocket.getOutputStream(), Constants.TELNET_ENCODING)); String firstLine; while ((firstLine = in .readLine())!= null)/*req from C*/ String res = kv.process(firstLine); out.write(res); /*to C*/out.flush();
```
C read resp.: /*after C connect S*/
```
String response = input.readline() ; s.close();
```
S shutdown: `ss.close();`



Finite state machines FSM describing a comm. session C S . C&S keep comm. session open and exchange Msg until one of them closed. accept: a while loop: create a new socket (and th.) for commu. with C



K V Server: C get, set Table in S: PUT tablename key value\r\n, GET tablename key |C: C.java:
```
KVTable<AssessmentInformation> kvc = new KVTable<~>(); kvc.put("middleware ", new AssessmentInformation(1.3);
```
KVTable.java in put():
```
String jsonStr = gs.toJson(value); String enc_value = encodeBase64(jsonStr); this. activeConnection.write("PUT "+ this .table + ""+ key + ""+ enc_value + "\r\n");
```
ActiveConnection.java in write():
```
PrintWriter output...; output.write(..); output.flush();
```
S: ConnectionHandleThread.java
```
BufferedReader in = new BufferedReader(new ISReader(clientSocket.getIS(),...) ); String line = in.readLine();
```
KVStore.java
```
StringTokenizer st = new StringTokenizer(firstLine, "\r\n "); String command = st.nextToken (); if (command == "GET")/*GET PUT else*/
```
ConnectionHandleThread.java
```
PrintWriter out = new PrintWriter( new OutputStreamWriter(clientSocket .getOutputStream(),...)); out.write( res);/*to C*/out.flush();
```

NIO(NonblockingSockets),multi.threaded

Synchronous: Single th. read data from C(stream) and blocked until done Asynchronous: th. → Channel: read data into Buffer; Channel → Buffer: fill data into Buffer; Th. → Buffer: check data in Buffer (main th. not blocked)

Syn. vs. Asyn.: Syn: Th. acts and waits until Syn. I/O done; textLimited scalability, a th. per I/O connection (Overhead:context switching → time between diff. tasks Asyn: Pass req. to OS-kernel and do other tasks →worker th.while(true) ++: only Compute, never Blocked, no Context Swtich, Asyn. I/O Scalability, Consumers not block S long time, One th. handle multi. sockets –: Complex handling code, Requires different kind of architecture(Eventloops)

Java NIO Channels: All I/O operations done with channels(File, TCP, UDP); Multi types of channels (FileChannel(File on disk), DatagramChannel (UDP), ServerSocketChannel, SocketChannel (TCP)); read/write), ServerSocketChannel (TCP)); Responsibility(Read, write buffer)

Sequence Dia multi th: n times C and Th. |Ci to S: connect , S to Th.i , Ci to S to Th.i: getTime() , Th.i to S to C: 20:00

S Create: same as single th. above S Create instead of block IO in while:
```
HandleRequest hr = new HandleRequest(client); hr.start();
```
HandleRequest as Th.:
```
class HandleRequest extends Thread{private Socket c; HandleRequest(Socket c){c = c;/*]*/ run(){String msg = c.receive();if(msg == "getTime"){String res = getTime(); c.send(res);c.close();
```
C:
```
class Client{public void sendMessage(){String ip='112.32.86.113 '; int port = 8883; String msg = " getTime"; Socket c = new Socket(); c.connect(ip, port); c.send(msg); String res = client.receive(); client .close(); print res;
```
Sequence Dia single th: n times C |Ci to S: connect , Ci to S to Th.i: getTime() , start with Zeitraum to process n req. Th.i to S to C: 20:00 resp. same order as sent req.

Selector: select multi asyn. Channels to match a EventLoop Setup channel:
```
Hashmap<SocketChannel, byte[]> pendingData; ServerSocketChannel ssc = new SSC(); ssc.connect(" 127.0.0.1", 8883); Selector selector = new Selector(); ssc.register( selector, SelectionKey.ACCEPT);
```
OR
```
ServerSocketChannel ssc = SSC.open ();ssc.configureBlocking(false); InetSocketAddress isa = new ISA( InetAddress.getByName("localhost" ), 5559);ssc.socket().bind(isa);
```
handleRequest():
```
public byte[] handleRequest(byte[] input){return Bytes.toBytes(System.nanoTime());
```
Eventloop:
```
public void start()throws IOException {while (true){try {selector.select();/*get List*/ Iterator selectedKeys = selector. selectedKeys().iterator(); while ( selectedKeys.hasNext()){SelectionKey key = (SelectionKey)selectedKeys.next (); selectedKeys.remove(); if (key. isAcceptable()){ServerSocketChannel channel = (SSC)key.channel(); SocketChannel socketChannel = channel.accept (); sC.configureBlocking(false ); sC.register(this.selector, SelectionKey.READ);/*wake up when sth to read*/else if (key.isReadable ()){SocketChannel channel = (SC)key. channel(); byte[] res = handleRequest (channel.read()); pendingData.add (channel, res); key.interesOps( SelectionKey.WRITE); else if ( key.isWritable()){SocketChannel channel = (SC)key.channel(); data = pendingData.get(channel); if( data = (byte[])"getTime"){channel .write((byte[])getTime()); key. interestOps(SelectionKey.READ);
catch (e){e.printStackTrace();
```
Read:
```
private void read(SelectionKey key) throws IOException {SocketChannel sC = (SC)key.channel();this.readBuffer .clear(); int numRead = sC.read( this.readBuffer);/*in Buffer*/if (numRead == -1){ key.channel() .close(); key.cancel(); return; /*]*/byte[] dataCopy = new byte[ numRead]; arraycopy(this.readBuffer .array(), 0, dataCopy, 0, numRead); handleRequest(sC, dataCopy);
```
Write:
```
private void write(SelectionKey key )throws IOException { SocketChannel sC = (SC)key.channel(); List queue = (List)this.pendingData.get(sC); while (!queue.isEmpty()){ ByteBuffer buf = (ByteBuffer)queue.get(0); socketChannel.write(buf); if (buf .remaining()> 0){break;/*]*/queue. remove(0);/*]*/if (queue.isEmpty()){ key.interestOps(SelectionKey.OP\_READ ); }/*]*/
```
Protocol Design: complex number as string: c = (a, b, n) , $n \in s,p$, $op \in \{add, sub, mul, div, poloar\}$, C to S req.: m<c1;c2;op>, <(1.0,1.0,s);(2.0,4.0,s);add> Status: $st \in \{OK, msgIncomplete.msgFormatError, serverError\}$, S to C resp.: m2<Cr; st>, <(3.0,5.0,s); ok> |Sequence Dia.: C2S: m1<(a1,b1);(a2,b2);".add» + S2C:m2<(a1+a2,b1+b2);".ok»; C2S: m1<(a1,b1);(a + m2<(0.0,0.0);"msgIncomplete» execution error, processing correct

3 External Data Representation

Presentation Layer ||Heterogeneity HW: Diff. HW architectures store bytes: Big, Small Endian ProgrammingLanguage:Diff. PL store data types diff.:AB0, 0AB, AB

Transformation between repre.s: Trans. between local and remote repre. (info may lost)

Two realizations: 1.Pairwise trans. between n local repre. vollständigGraph,$\#n^2 - n$, Either sender or receiver has to transform 2.Trans. to and from canonical repre. single canonical C as middle repre., No local info about comm. partner; $\#2 * (n - 2)$ |−2f canonical is one of n

XDR OSI Pre. Layer, encodes only data items, no meta info(types) ++: easy, Fixed length reduces computation –: Receiver need data description, wasting, no meta data |exactly 4 Byte(32 bit) int using big endian → n bytes data; r bytes padding; $(n + r)\ mod 4 = 0$ |int32; float=Sign+Exponent+Mantissa, String=length_int32+bytes, array=length_int32+ele
```
struct forecast {String weekday;int temperature; String tags<>;//]}
```
weekday: 00 00 00 06, 6D 6F 6E 64, 61 79 00 00 → "mond ay__" ; temperature: 00 00 00 0E ; tags: 00 00 00 02, 00 00 00 05, 73 75 6E 6E, 79 00 00 00, 00 00 00 03, 64 72 79 00→ sunn y___ dry_" Total:40 B

ASN.1: Abstract description of data types encoding(telecommunication,internet protocol); Enables exchange in heterogeneous SY; ++: receiver not need to know data description –: additional overhead(types) abstract Syntax *compiler* → concrete Syntax: Java, C++, they transfer syntax using encodingRules |data: Tag: 2+1+5=8bit, Class: 00 UNIVERSAL,01 APP,10 Context,11 PRIVATE, f0 primitive,1 constructed, numbers:from table + Length 8bit + Value multi 8bit in Type Table: decimal, in Encoding: 0x
```
Forecast::=SET{weekday IA5String, temperature Integer,tags SEQUENCE OF IS5String;//]
```
weekday: 16, 06, 6D 6F 6E 64 61 79 → (16)0x for 22 for IA5String ; temperature: 02, 01, 0E → 02 for int ; tags: 30, 0C, 16, 05, 73 75 6E 6E 79, 16, 03, 64 72 79 → 30 for SEQUENCE_OF, 12 for all Nums Total:27 B

Java object serializationJOS Stream-based transmission of serialized objects(Via TCP or UDP sockets); Receiver needs implementation of class; Serial. not require class specific code(Java reflection); java.io.Serializable |`public class Person implements Serializable{String name;` –: locked into Java(No support for heterogeneous systems); No support for versioning (If the serialized class changed eg. new attribute added, update all network nodes) serialize:obj2bit
```
try{ Socket s = new S("localhost", 8022) ;ObjectOutputStream oos = new OOS(s .getOutputStream()); oos.writeObject (obj);/*write obj stream to socket */catch (e){e.printStackTrace();
```
deserialize:bit2obj
```
ServerSocket ss = new SS(8022);Socket s = ss.accept ();ObjectInputStream ois = new ObjectInputStream(s.getInputStream()) ;obj=(Obj)ois.readObject();
```
XML De facto standard XML Schema:
```
<xsd:element name="forecast"> <xsd:complexType> <xsd:all> <xsd:element name="weekday"type="xsd:string" /> <xsd:element name="temperature" type="xsd:integer"/> <xsd:element name="tags"><xsd:complexType> <xsd: sequence> <xsd:element name="tag"type ="xsd:string"maxOccurs="unbounded \"/> </xsd:...>
```
XML Document: 116B:
```
<forecast> <weekday>monday</weekday> <temperature>14</temperature> <tags> <tag>sunny</tag> <tag>dry</tag> </..>
```
XML RPC Interoperable RPC standard, Predecessor of SOAP, light-weight, HTTP

JSON human readable text to transmit data objects; JSON databases: MongoDB JSON obj:
```
{"forecast":{"weekday":"monday"," temperature":14,"tags":["sunny"," dry"]//]}
```
++XML/JSON: readable, defined as standard, JS support JSON(directly loaded Browser and deserialized), XML Builders/Parsers –XML/JSON: verbose, bad Preformance, Overhead(XML), more than binary data repre., slow Write&Parse JSON RPC: Stateless, light-weight, avoid complexity like XML-RPC using JSON, many prog. lang.(JS, C, C++, Java, PHP) Req. batch:(protocol version 2.0, methodName req.id)
```
[{"jsonrpc": "2.0", "method": "sum ", "params": [1,2,4], "id": "1"/*] */, {"jsonrpc": "2.0", "method": " notify\_hello", "params": [7]/*]*/, { "jsonrpc": "2.0", "method": "foo.get" , "params": {"name": "myself"/*]*/, " "id": "5"/*]*/, {"jsonrpc": "2.0", " method": "get\_data", "id": "9"/*]*/]
```
Resp. Batch:(result,req.id)
```
[{"jsonrpc": "2.0 ", "result": 7, "id": "1"/*]*/, {" jsonrpc": "2.0", "error": {"code": -32600, "message": "Invalid Request "/*]*/, "id": null/*]*/, {"jsonrpc" : "2.0", "error": {"code": -32601, " message": "Method not found"/*]*/, " id": "5"/*]*/, {"jsonrpc": "2.0", " result": ["hello", 5], "id": "9"/*]*/ ]
```
ProtocolBuffers PB: Google Concept like ASN.1 , but not standard, efficient binary serialization, heterogeneous SY(Java, C++, C#) ++: efficient write/parse, Well documented, Versioning, marshalled efficiently, faster than Thrift, data compression –: No RPC, binary format not readable |data structures independently defined in .proto file(IDL) → generate serialization code(Java, C#) → Java, .NET projects request, response each other DataModel.proto:
```
message forecast{ required string weekday =1;required int32 temperature =2;repeated string tags =3; //optional float price=4;]
```
fieldTag(5bit)=1, type(3bit)=2(lengthDelimited: string) :00001 010 → OA, 06(length), 6D 6F 6E 64 61 79 ("monday", utf8, 0x); fieldTag=2,type=0(Var-int) :00010 000 → 10, 0E; FieldTag=3,type=2 :00011 010 → 1A, 05, 73 75 6E 6E 79 (sunny"); FieldTag=3,type=2 :00011 010 → 1A, 03, 64 72 79 ("dry") RPC PB/Thrift: each call define Msg type .proto(compile in C++)\.thrift file C → C Stub(Serializer, Transport) → LAN\Internet → S Stub(Deserializer(generated Code), Transport(PB) \TSocket(Thrift library)) → S Process: 1.Define the RPC service interface in a proto \thrift file 2.Generate code 3.Implement service 4.Implement C S (Thrift-library:Tsocket, TServerSocket, Tserver, TSimpleServer) RPC ServiceInterface.proto:
```
enum Status{OK=0; EXISTING=1;NOT\_EX=2;ERROR=3;//] message SearchRequest{required String attribute=1;required String value=2;//] message CustomerList{ repeated Customer customers=1;//] service AdministrationService{rpc CreateCustomer(Customer)return( Status); rpc DeleteCustomer(Customer )return(Status); rpc SearchCustomer( SearchRequest)return(CustomerList);// ]
```
RPC Thrift(code in Thrift S): Server(APPLayer)= Processor(ServiceLayer) + Protocol(Pres.Layer) + Transport(CommLayer) Thrift Service:
```
service MathematicsService{i32 add(1:i32 add1 , 2:i32 add2)//]
```
Java Impl. Service:
```
public class MathematicsServiceHandler implements MathematicsService.Iface {public long add(int add1,int add2) throws TException{return add1 + add2;
```
Thrift S:
```
public class MathematicsServer {public void start(Integer port)throw TTransportException{TServerSocket serverTransport=new TSS(port); MathematicsService.Processor p=new MS.P(new MathematicsServerHandler() ); TServer s=new TThreadPoolServer (new TThreadPoolServer.Args( serverTransport).processor(p)); print ("StartingS"); server.serve();
```
Thrift C:
```
public class MathematicClient { public Integer add(Integer add1, Integer add2){TTransport transport ; try {transport = new TSocket( mIpServer, mPort); TProtocol protocol = new TBinaryProtocol(transport); MathematicsService.Client c=new MathematicsService.Client(protocol); transport.open(); Integer addResult=c .add(add1, add2); return addResult; catch (TTransportException TException ){finally {transport.close();
```
Apache Thrift: framwork, applied Hadoop and HBase Workaround like Protocol Buffers, but starting with .thrift file ++: Multi protocols to serve diff. purposes(binary, JSON), RPC library, RPC transport Open source, widely used, Versioning, Streaming, Compile to multi PL(C#,PHP,Ruby), data structures(Map, Set, List, Excep), RPC implementation –: No Dyamic Binding .thrift file:
```
struct Forecast{1: string weekday 2: i32 temperature 3: list<string> tags//]
```
fieldTag(4bit)=1, type(4bit)=8(string) :0001 1000 → 18, 06(length), 6D 6F 6E 64 61 79 ("monday", utf8, 0x); fieldTag=2,type=5(int32) :0010 0101 → 25, 0E; FieldTag=3,type=9(list) :0011 1001 → 39, 28(0010(2 items→2) 1000(string→8)), 05, 73 75 6E 6E 79(sunny") 03, 64 72 70("dry"), 00

Variable Num encoding Protobuf Protocol: last 8 bit with most significant bit 0 fieldTag(5bit)=1, type=0(Var-int) :00001 000 → 08, d0(11010000) 91(10010001) 32(00110010) → 821456; fieldTag(5bit)=1, type=0(Var-int) :00001 000 → 08, 9f(10011111) a3(10100011) 64(01100100) → -821456 ZigZag:1642911=|-821456|*2 - 1;

4 RPC/SessionLayer

Remote Procedure Call: Call non-local procedure on remote S, like local call |Hidden: Msg Def., content types, Marshalling/unmarshalling of parameters, Sending/Receiving Msg

Sequence流程, COMU 1. C call RPC issuing req. to C stub (a proxy obj). 2. C stub encodes para.(marshalling, Chap2) 3. Network Address(LAN, Internet) given, C stub calls the S stub via RPC, which executed by RPC library, RPC protocol ,socket connection. 4. S stub receives RPC. 5. S sub decodes para.(ummarshalling) and calls local procedure at S. 6. S executes RPC. 7. Local procedure $\xrightarrow{result}$ S stub $\xrightarrow{result}$ C.

RPC Parameter CallByValue: easy CallByRef: not work(C,S different address spaces) → CallByCopy (serialize objects with Thrift, Protocol Buffers)

RPC Binding StaticB: Hardcoded reference to S; ++: Simple, Effective, No Additional Infrastructure –:C S tightly coupled DynamicB: Relies on Name and directory services to locate S; Cost in additional infrastructure, protocol, registration primitives; → In Sequence Above流程 Statt 3. C requests service, executes RPC with S address. NameService provides S address (S registed service at directory before)

RPC with Sockets In Sequence Above流程 Statt 3: RPC call, P2P comm, C Socket(added under RPC Protocol) connect S port(S Socket and Socket). S accept connection.

RPC Errors 1.Lost Req. longer than Timeout of C, C resend with SequenceNum → No Duplicates at S Send fail, Resend before req. done, Resend before req. comes 2.Lost Reply Same resend after Timeout S caches result, repeat result back when Duplicates coming. If reply acknowledged, delete cache Reply fail, Reply longer than Timeout 3.C Crash S waits ACK forever(orphan). C restart using SessionCounter → old reply not mix with new requests C Crash sofort after req., but first resp. after restart(not mix) 4.S Crash C not Know RPC executed or not. S logs keeps state of the procedure → no Multi Executions state: how many % of resp. done

Failure Semantics 1.Maybe No repeat, One Try, No guarantee for success ++: Simple, Efficient, 2.AtLeastOnce Repeat after Timeout until ACK, No ID of duplicates , Acceptable for idempotent(always same result) operations 3.AtMostOnce Duplicate不重复处理，不重复回复 ID of duplicates: sequenceNum, Repeat resend, but No Repeat Execution , Acceptable for non-idempotent operations , No (or unknown result) when S crash Executed也不保存 4.ExactlyOnce Duplicate不重复处理，不重复回复 State of procedure recorded, Only possible with TX → Atomic execution, After S Crash and C resend, the crashed operation recovered and executed exactly once

Syn. Asyn. RPC Syn. RPC: C blocked until result from S comes Asyn. RPC: lets C continue Asyn. C: handle syn. to call caller th., Reply of RPC provided by executing a callback from caller th. to C or C call polling to caller th.. Asyn. C: C calls redirected by S to multiple handler th.(th. pool) |||RPC Problems: not like local procedure call: error semantics, Reliability depends on underlying protocol (TPC), Separate address spaces, Lack of shared variables + Performance Transparency + Scalability + Security

Remote Method InvocationRMI Object-oriented RPC; Procedure call on remote Obj; Method para. be send two ways (callByValue→Implement Serializable, callByReference→extend Remote interface); Obj reference identifies remote Obj. ||RPC vs RMI RPC Procedures of remote processes called, Service interface provides set of procedures RMI Obj in diff processes comm. with each other, Remote interface specifies methods of obj |RMI Protocol: C → Stub(Remot Ref. Layer, Transport) → LAN\Internet → Skeleton(Remot Ref. Layer, Transport, RMI Registery) → S |RMI technology: CORBA, COM DCOM .NET, Java RMI: Define ,extend remote interface method:
```
import java .rmi.Remote+RemoteException;public interface Compute extends Remote{< T> T executeTask(Task<T> t)throws RemoteException;
```
Define para.:
```
import java.io.Serializable+math.BigDecimal ;public class Pi implements Task< BigDecimal>,Serializable{private static final long serialVersionUID =227L;public BigDecimal execute (){return computePi(digits);/*]*/ public static BigDecimal computePi (int digits)
```
Implement remote obj:
```
public class ComputeEngine implements Compute{public ComputeEngine(){super ();/*]*/public <T> T executeTask( Task<T> t){return t.execute();/*]*/ public static void main(String[] args ){if(System.getSecurityManager()== null){System.setSecurityManager(new SecurityManager());/*]*/try{String name="Compute"; Compute engine=new ComputeEngine(); Compute stub=(C )UnicastRemoteObject.exportObject (engine,0); Registry registry= LocateRegistry.getRegistry(); registry.rebind(name,stub); print ("ComputeEngine bound"); catch e
```
Register object at S,Lookup and C call obj:
```
public class ComputePi{public static void main(String args[]){if(System .getSecurityManager()==null){System. setSecurityManager(newSecurityManager ());/*]*/try{ String name="Compute" ; Registry registry=LocateRegistry .getRegistry(args[0]); Compute comp =(Compute)registry.lookup(name); Pi task=new Pi(Integer.parseInt(args[1]) ); BigDecimal pi=comp.executeTask( task); print(pi); catch e;
```
–: Locked into Java, not scale well, Versioning problem, Average performance, latency insufficient No global obj registry

More to Sec2. Protocol Design:
```
class Complex{double a,b; /*Cons.*/ public String marshal(){return concatStrings('(',valueOf(a),',', valueOf(b),')';/*]*/public void unmarshal(String complexString){a= parseDouble(removeString(splitString (complexString,',')[0],'(')); b= parseDouble(removeString(splitString (complexString,',')[1],')'));/*]*/
```
M1:
```
class M1{Complex c1=new;Complex c2=new;String op='+';/*Cons*/public String marshal(){return concatStrings ('<',c1.marshal(),';',c2.marshal(),'; ',op,'>');/*]*/public void unmarshal (String m1String){c1.unmarshal( removeString(splitString(m1String,';' )[0],'<')); c2.unmarshal(splitString (m1String,';')[1]); op=removeString (splitString(m1String,';')[2],'>' );/*]*/
```
M2: one Complex, String st='ok'.

RPC C:
```
class Client{/*App.Layer*/ public Complex callRemoteService( Complex c1, Complex c2, String op){ M1 m1 = new; M2 result=sendRPC(m1); return result.cr;/*]*//*Repre.Layer */public M2 sendRPC(M1 m1){String result=sendMessage(m1.marshal()) ; M2 m2=new; return m2.unmarshal( result);/*]*//*SessionLayer*/public String sendMessage(String message ){String ipAddress='112.32.86.113 '; int port=8883; Socket c=new; c .connect(ipAddress, port); c.send( message); String result=c.receive (); c.close(); return result;/*]*/
```
RPC S:
```
/*APPLayer*/public Complex callLocalService(Complex c1,Complex c2,String op){Complex result; switch (op){case "+":return new Complex( c1.a + c2.a,c1.b + c2.b); case "-" :return new Complex(c1.a - c2.a,c1 .b - c1.b); case "*": .../*]]*//* Repre.Layer*/public void processRPC( String rpc){M1 m1 = new; m1.unmarshal (RPC); Complex cr=callLocalService (m1.c1,m1.c2,m1.op); M2 m2=newM2( cr,'ok'); return m2.marshal();/*] *//*SessionLayer*/provideService(){ int port=8888; ServerSocket ss=new ; while(true){Socket c = ss.accept (); String message = c.receive(); result = processRPC(message); c.send (result); c.close();/*]]*/
```
Values never has delimiter; If class changed, parsing the old classes fails; Obj meta info stored into Msg→XML; Obj properties serialized in bytes, fixed length intervals→Protocol Buffers

5 Naming, Directory

Naming Service Collections(Naming Contexts) of <Name, Attribute> Pairs, resolve name to physical address, Decouple logical& physical concepts. Use Case: ID of address or attribute for nameDNS, ID of machine for serviceRPC, ID of machine for objRMI.

DNSDomain Name SY → url to IP of host serving this domain. Map domain names to IP, worldwide distributed DB of name servers |Types of Domain Names: Root domain: top of tree unnamed level(.) Top level do.: country/region or organization .de Second level do. Variable-length names organization tum.de Subdomain: Additional namespace in.tum.de Host or resource name leaf in DNS tree, a specific resource host-a.in.tum.de DNS follow this top down .de → server.in.tum.de |Name S,DNS S: Caches result, Authoritative name S responsible for domain, each domain has >= one, Maintains list DNS records in zone file Non-authoritative name S Receives info about domains from other S Forwards queries, answers with cached result. |Name resolution types: Recursion: C DNS req.1 ↔ DNS S1 req.2,resp.4 ↔ DNS S2 resp.3 Iteration: C DNS req.1 ↔ DNS S1 add from S2, C DNS req.3 ↔ DNS

S2 DNS resp.

Directory services Shared info SY infrastructure in DB, Serve to locate, manage, administer, and organizing networked resources, Look up entries match attribute. vs DB: Directory info read often than write; Data stored redundantly better performance; S Distributed hierarchical. Naming vs directory services: white page, lookup phone number by name Map<String, Attr> VS yellow page,look up by attributes Map<String, Map<Attr>> ||LDAP: Lightweight directory access protocol, directory information tree DIT, node: directory information base DIB. Root→de→structure net→Sales\Dev.\Support →Employee, Operations: Add(to DIT) ,Bind ,Delete(only leaf), Search and compare, Modify, Modify DN UDDI: Universal Description, Discovery and Integration Functions: White pages: Address, contact, and known identifiers; Yellow pages: Industrial categorizations based on standard taxonomies; Green pages: Technical info about services and reference to WSDL description JNDI: Java Naming and Directory Interface, Allows Java clients to discover and look up objects via name, service provider interface

6 WebService

SOAP Simple Object Access Protocol Msg format: Envelope Enclosing entity of Msg, Defines namespace Header Contains metadata for the body, Many WS-* extensions Body Contains the payload, Further specifications define body structure RPC vs DocumentStyle SOAP Msg constructed in a specific way, call the Web service, Body of Msg contains para. and method name as wrapper element, marshalling/unmarshalling RESTful: Representational State Transfer, architectural style distr. SY |contains no restrictions, Message body is a XML document, C/S handles the marshalling/unmarshalling XML Schema Data Model for web service:

```
<schema targetNamespace="http://company.com/datamodel.xsd"xmlns="http://www.w3.org/2000/10/XMLSchema"xmlns:dm="http://company.com/datamodel.xsd"><complexType name="customerType"><all><element name="customerId"type="integer"/><element name="date"type="date"/><element name="name"type="string"/><element name="address"type="dm:addressType"/></all></complexType><complexType name="addressType"><all><element name="city"type="string"/></all></complexType><complexType name="orderType"><all><element name="items"><complexType><sequence><element name="item"maxOccurs="unbounded"type="dm:itemType"/></...>
```

WSDL file for web service interface:

```
<definitions name="CustomerService"targetNamespace="http://company.com/customerservice.wsdl"xmlns:ws="http://company.com/customerservice.wsdl"xmlns:dm="http://company.com/datamodel.xsd"xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"xmlns="http://schemas.xmlsoap.org/wsdl/"><message name="AddItemToOrderInput"><part name="orderId"element="xsd:integer"/><part name="item"element="dm:itemType"/></message><message name="AddItemToOrderOutput"><part name="order"element="xsd:boolean"/></portType name="CustomerServicePortType"><operation name="AddItemToOrder"><input message="ws:AddItemToOrderInput"/><output message="ws:AddItemToOrderOutput"/></><binding name="CustomerServiceSoapBinding"type="ws:CustomerServicePortType"><soap:binding style="wsdl:document" transport="http://schemas.xmlsoap.org/soap/http"/></><service name="CustomerService"><documentation>Customer Service</documentation><port name="CustomerServicePort"binding="cs:CustomerServiceSoapBinding"><soap:address location="http://company.de/customerservice"/>
```

7 Messaging&Queuing

Msg Queuing Pattern data shared across APP diff. platforms, MQP transfer data frequent, immediate, reliably, exactly once, asy. Enterprise APP integration EAIAPP comm. using single Msg Bus

Msg Passing: Msg=receiver+sender+type+payload send(message); receive(message); callback(); Messaging vs RMI interface (per Msg product) with generic operations \required and known, but differs across apps; Lower-level of abstraction than RMI \like non-remote calls but fail potential; Flexible in that messaging allows arbitrary interaction patterns between sender and receiver; Sender not blocked \blocked until reply arrives after sending; emulate \Realize request/reply pattern; Asynchronous behavior hard to use debug |||Msg Passing Properties 1.Reliability SequenceNum, Ack, Timeout resend →Msg loss; CheckSums, Error correcting codes (redundancy), Repeat send →Msg Corruption; Loss+Corruption+Msg encrypting →Malicious Intend 2.Ordering Msg arrive in order: FIFO(for one sender),Causal,Total(for all sender); GOAL: All Msg delivery OR Fast incomplete Msg delivery 3.Syn vs. Asyn send\receive: 4.Buffering Msg Queuing Transient vs Persistent Msg: Msg not stored VS Msg stored lifetime of sender&receiver until revoked, purged, queuing, Msg queue |Blocking Send&Receive Semantics Blocks send call until Msg arrived: local middleware \ remote middleware \ receiving process \ processed by receiving process; Blocks receive call until: Msg available use non-blocking polling \ Timeout \ Msg message received |Syn.Comm. Sender blocked until Re. done, Receiver blocked until Se. done, Syn equalizes Se.Re. speed → Deadlock(both send wait), ++: Se. knows Msg was received, Only one Msg buffered, Synchronizes Se. and Re. operating diff. speeds –: coupled (at same time), blocked(deadlock, no paral.) Asy.Comm. Send not blocked; MW buffers Msg on Se.\Re., SenderBuffer send faster → transmission, ReceiverBuffer receive more Msg > processing, → Buffers temporarily(not permanently) mitigate speed difference ++: ring buffer, loosely coupled, not same time, parallelism –: Se. know Msg received, Buffers overflow, Dev. Debug. complex Msg Buffering send buffer full → Se. block, buffered MsgsOver-written, Return Exception, error |receive buffer full → Msg dropped, buffered Msg over-written, return NACK to MW at S. Sliding Window Protocol TCP prevent Re. buffer overflow: send side may only have number of Un-ACK Msg (sliding window): Re. send ACK, SW size determined statically or dynamically → size < buffer size (nooverflows), size > buffer size(high throughput but can overflow flow control (dynamically adjusting the window size) Syn Comm via Asy MW: Se. wait for Re. ACK

Queuing Msg passing+Msg buffering Dequeue: Remove: remove Msg if dequeued Browse: still available for other Re. if dequeued Decoupling Ser,Rer → indirect and asy comm.(asyn received via Callback = deque) Space de.Se not know Re, but know Queue, Time de.Se Re not same time, Flow de.Sending not blocked→non-blocking enqueue Msg Queuing Pattern: OneToOne, OneToMany, ManyToOne, ManyToMany Queue Manager: Specialized component to queuing functionality to apps administrative functions(create, delect, start, stop of queue) Multiple Consumer Queue Msg Property: Priority: Static: High first OR Weighted fair scheduling:交换a:weight2 abacabac Dynamic: Priority change, Earliest deadline first EDF, Least laxity first LLF(min time left to DDL) Earliest dequeue time: before it Msg not dequeued Expiration time: Msg deleted if expires Correlation identifier: match req with reply TX Queue fail recovery: ACID atomic consistent isolated durable; Enable local Msg TX; commits succeeds or aborts fails; Msg received/sent if commits; Producer side Msg kept until commit, If aborts, Msg discarded; Consumer side All consumed Msg kept until commit, If aborts, re-delivered; Request/Reply Queue,not atomic Req with correlationID to match reply at C C 1.Req Enque → Req.Que 2.Req.Deq → S 3.Req.process Rep.Enque *rightarrow* Rep.Que 4.Rep.Deq → C Request/Reply TX Queue local TX: 1.C RequestTX: C req 2.S TX: S deq req, processes, enq reply 3.C Reply TX: C deq. reply C RequestTX committed: Req. process exactlyOnce, Reply process atLeastOnce R\R TX workaround: C ReqTX |Start enqueueRequest1.enq. Commit| → Req.Que → S TX|Start dequeueRequest1 enqueueReply2.enq. Commit| → Rep.Que → C RepTX|Start dequeueReply2 Commit| |C checks queues Req. in req. que. OR Req. currently processed by S OR Reply in reply que. Case of Abort C Request TX abort, req enq. undone; STX abort, req deq.& reply enq undone; C Reply TX abort, reply deq undone

C crash, Recovery Persistent at C and queue manager: C marks req with ID, QM store IDs of the last enqueued request Re, the last dequeued reply Rd, updated after commits. R:ID of most recent request by the C (stored by C) 1.C ReqTX not commit $R \neq Re$, resend, Empty queues, old Re in QM 2.C ReqTX committed but S TX not$Re = R \neq Rd$ Req in ReqQue(req.que.size!=0,dequeue) OR being processed by S(req.que.size=0,C wait) 3.S TX committed but C RepTX not Reply in Rep.Que(nonempty, $R = Re$, QM's C's ReplyID un-match) , 4.C RepTX committed $R = Rd$ Successful,continue

Distributed TXQueuing Two-Phase Commit 2PC: Only Atomicity, Coordinator among registered nodes to achieve consensus; Coordinator requests all nodes to vote; commit, if all voted to commit; abort, if one voted to abort; 1.Prepare to commit, voting phase Once a participant voted to commit, no longer abort the TX 2.Commit completion phase; commit, after all participants prepared Otherwise all abort

Class C S ChannelFactory cf=new();Producer p;Consumer c; S C init(): cf.setHost("broker.tum.de");cf.queueDeclare("req_q", Persistent);cf.queueDeclare("res_q", Persistent);p=cf.newProducer("req_q");c=cf.newConsumer("res_q");//for S change p and c, C S different init() R\R Queue C send(byte[] msg): p.enqueue(msg); C byte[] receive(): return c.dequeue(); S run(): while(true)byte[] req=c.dequeue();try Database.store(req);p.enqueue('ACK'.toBytes())catch(e)p.enqueue('ERROR'.toBytes()); Correlation: C: + Hashmap<Guid,byte[]> intermediate=new();Hashmap<Guid,Correlation>correlations=new();//two also for Broker C run(): Thread.start(new Thread(){void run(){while(true){Delivery d=c.dequeue();/*stat in broker responseConsumer.dequeue()*/if(d!=null){Guid g=d.getGuid();if(intermediate.containsKey(g){Correlation cor=new(g,intermediate.get(g),d.getMessage());intermediate.remove(g);correlations.add(g,cor);/*statt in Broker: clientQueues.get(guid.getClientName()).enqueue(g,cor.toBytes());*/ C send(byte[] msg): Guid g=Guid.generate();p.enqueue(g,msg);intermediate.add(g,msg);//not in Broker C S run(): while(true){Delivery req=c.dequeue();try{Database.store(req);p.enqueue(req.getGuid(),"ACK".toBytes());catch(e)p.enqueue(req.getGuid(),"ERROR".toBytes()); Queue with Broker: C: + String clientName C init(): + /*producer*/cf.queueDeclare('req_que_broker',Persistent);cf.queueDeclare('c_que_'+clientName, Persistent);/*consumer*/ C void send(): see Correlation part above C void receive(): Delivery d=c.dequeue();if(d!=null){Guid g=d.getGuid();Correlation cor=new();cor.fromBytes(d.getMessage()); Broker Class: + List<String> clients=new ArrayList<>();List<String> servers=new ArrayList<>();Map<String,Producer> serverQueues=new HashMap<>();Map<String, Producer> clientQueues=new HashMap<>();Consumer requestConsumer, responseC;//CF,Host,2 HashMap like Correlation C Broker init(): clients.add("c1");servers.add("s1");/*multi add*/cf.queueDeclare("request_queue_broker",Persistent);requestConsumer=cf.newConsumer("response_queue_broker");cf.queueDeclare("response_queue_broker",Persistent);responseConsumer=cf.newConsumer("response_queue_broker");for(String s:servers)/*also for client*/String queueName="server_queue_"+s;/*client_q*/cf.queueDeclare(queueName,Persistent);serverQueues.put(s, cf.newProducer(queueName));//clineQueues, repeat this for loop for Client Broker run(): Thread.start(new Thread(){void run(){int count = 0;while(true){Delivery d=requestConsumer.dequeue();if(d!=null){intermediate.add(d.getGuid(),d.getMessage());serverQueues.get(count%servers.size()).enqueue(d.getGuid(),d.getMessage());count++; //HERE: REUSE Correlation: Client run()function S: + String serverName S run(): while(true){Delivery d=c.dequeue(try{Database.store(request);p.enqueue(d.getGuid(),"ACK".toBytes());catch(e)p.enqueue(d.getGuid(),"ERROR".toBytes()); TX Queue + C: Queue<byte[]>messageBuffer=new;Queue<Delivery>resultBuffer=new; C request(): guid=Guid.generateRandom();f=newFile("path");f.write(guid);startTX message=messageBuffer.get();producer.enqueue(guid,message); commitTX C reply(): startTX Delivery d=c.dequeue();resultBuffer.put(d); commitTX; C recovery(): f=newFile("requestFile");Guid R=f.readLastEntry(); Guid Re=cf.getLastEnqueued("request_queue");Guid Rd=cf.getLastDequeued("reply_queue");if(R!=Re) requestTX() else if(R!=Rd){while(cf.getQueueSize("reply_queue")==0)Thread.sleep(500);/*wait all out*/replyTX()else/*nothing*/spawn(while(true)) S: processTX startTX Delivery d=c.dequeue();p.enqueue(d.getCorrId(),"ACK"); commitTX;

8 7QueuingTheory

ArrivalRate$\lambda = \frac{1}{meaninter-arrivaltime}$ |ServiceRate $\mu = \frac{1}{meanservicetime}$ |Q.Formation $\equiv \lambda > \mu$ |Stability $\equiv \lambda < \mu$: busy,idle alternating |Throughput:average # completed jobs per unit of time → 0(stable) to maximumThr.(unstableQ.,best performance) |ServerUtilization $= \lambda * \mu$: S busy time percent |A/S/n: arrival process/ service process/# S |A,S:M (Markov,Exponential probability density), D (Deterministic, All customers have the same value), G (General, Any arbitrary probability distribution) |M/M/1: Infinite population of customers, Infinite q.capacity, FIFO |Exponential distribution: $f(x) = \lambda e^{-\lambda x}$, mean= $1/\lambda$ |Poisson arrival process(arrivalProcess): $P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$, |Traffic intensity (occupancy): $\rho = \lambda/\mu$, n customers: $Prob[n] = \rho^n(1-\rho)$, empty: $Prob[n = 0] = 1-\rho$ |Mean # customers: $N = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda} = L$ |Total waiting time (including service time): $T = \frac{1}{\mu-\rho}$

9 S-Side Architecture

Thread: has own Stack, Instruction pointer,Processor State(Register). multiple included in proc. |**Process**:running program as set of Threads(Binary image loaded into memory, Instance of virtualized memory, Resources as open files, sockets, Security context as associated user, Thread as Unit of activity of process) |**Single-threaded Pro.** One instance of virtual memory |**Multi-threaded Pro.**: One instance + th. share same memory address space(ob) |**Synchronisation**:sharing same address(obj), Read/Write access must be synchronized(Mutex, Locks, Semaphores) Lock:`Lock l; l.lock(); /*critical section, can be Deadlock*/l.unlock();` Mutal Exclusion:Critical sections of th.s not overlap Freedom from deadlock, Freedom from starvation |**Th. State** Created → Waiting(Scheduled or unscheduled) ↔ Swapped outWaiting(No enough memory), Running → Terminiated, Blocked(Blocking IO) → Waiting(IO finish) Blocked ↔ Swapped out andBlocked(No enough memory) |**API**: `public class WorkThread extends Thread{ public ConnectionHandleThread(Parameter params) @Overridepublic void run(){ Computation(); /*usage*/Thread t = new WorkThread(params); th.start(); Thread.sleep(1000);/*current thread */th.getState();/*NEW, RUNNABLE , BLOCKED, WAITING, TERMINATED, TIMED_WAITING*/th.setPriority(para); th.yield();/*give current use of processor to other th.*/` |**Amdahl's law**: maxium theoretical speedup using multiple processors $T(n) = T(1) * (B + \frac{1}{n}(1-B))$ n:# processor, $B \in [0,1] serial, nonparallizable$, speedup:$S(n) = \frac{T(1)}{T(n)} = \frac{1}{T(n)}$ |**Overload**: 1.Transient ov.(Short spikes in requests which ov. the S temporarily) 2.Continous ov.(Incomming rate exceeds the capacity of the S) 3.Methods to tackle ov.(Refuse requests after certain queue length,Add capacity) |**Concurrency Models** 1.**Thread-based conc.** (One request is completely handled by a single th. time of request, same th. is responsible until the response is returned If request is blocked (due to a DB access), the th. is blocked 2.**Event-based conc.**not th. based Request in Network/HDD→EventHandler→# FSM = # Steps(th.s)→Response (–: Only non-blocking(async) I/O operations used, Code less modular,nonreusable, Complex control flow(FSM, flags), debugging OS support like async I/O libraries) (A th. processes a request until th. blocked. When th. blocked, the th. continues with the next requests, Once a th. becomes unblocked, it gets enqueued in the list of ready tasks. Scalar code separated in micro tasks. Finate State MachinesFSM(Java NIO)to track state) C socket created → th1 accept connection register 'C socket event' |requests from C socket → th2 lookupDB register 'query done event' |QueryDone → th3 send 1MB to client register 'data sent event' |data sent → th4 Cancel connection |**Single-threaded S = Iterative S**: S only single worker th.(do all also I/O), Thread-based conc. Waits till a new request from Network is in input queue, if empty ,Dequeues the next request ,Computes the result of the request ,Generates response to Network ,When blocked due to I/O, thread also blocks(I/O read disk to memory) ++:Simple programming, cache locality(one request in cache) –:Insufficient use of resources, Limited scalability(reject requests if too many) |**Multi-threaded S**: S with multiple worker th.(same way as single th. wait . . . block OS can schedule non-blocked threads(get nonblocked th. from pool), more threads than cores since a certain percentage of I/O is assumed ++: Programming abstraction(Dividing up work and assigning to a th.), Parallelism(better throughput), Responsiveness(Doing heavy operations in background, while UI stays responsive), Blocking I/O(When blocking I/O, other th. continue to work), Contex switching(Switching costs from one thread to another within the same process is cheaper than process-to-process context switching), Memory savings(Th. to share memory, yet utilize multiple units of execution) |**One th. per request/Connection**:scheduled by operations, Request in Network→InputQueue→Dispatcher (starts one th. per request,when response sent, th. ends)→Response to Network ++:Simplified programming –: Too many th. → thread-starvation(too little CPU-time) , cost of Starting/Removing th. , Not ideal for highly parallel S `ServerSocket ss = new ServerSocket(); ss.bind(new InetSocketAddress("127.0.0.1", port)); while (true) Socket ss = serverSocket.accept(); Thread t= new ConnectionHandleThread(kv, ss); t.start();` + `public class ConnectionHandleThread extends Thread { public ConnectionHandleThread(KVStore store, Socket clientSocket){... @Override public void run(){ BufferedReader in = ... clientSocket. getInputStream(); PrintWriter out = ... clientSocket.getOutputStream(); String firstLine; while ((firstLine = in.readLine())!= null){String res = kv.process(firstLine);out.write(res); out.flush();` |**Abstracting Th.**: deadlock free, starvation free **Abstractions:** Thread pool, Event based concurrency, SEDA, Actor model of concurrency, Reactive programming |**Th. pool**managed collection of available th.s, size as maximum(eg number of CPU cores), Completes tasks in parallel, Reuses th. for multiple tasks **Multi-threading with th. pool** optimal size depends on # cores, # blocking of I/O operations |Request in Network→InputQueue→Scheduler(Thread pool with size)→Response to Network `ExecutorService es= Executors .newFixedThreadPool(4);while (true){Socket serverSocket= serverSocket.accept();es.submit( new ConnectionHandlerRunnable(kv, clientSocket));` ++ RUNNABLE above in One th. per request/Connection |**Scheduling Algorithm**:Most cache efficient to assign tasks to th.s in pool, Two main paradigms:Work sharing,Work stealing Work Sharing Algo.When th. created, scheduler moves work of other th. to new th. –:Comm. between cores, Cache misses |Work stealing algo. Under-utilized processors steal work from busy processors(The migration of threads occurs less frequently with work stealing than with work sharing) ++:Maintain th. on same CPU(data locality → better cache usage, Minimize comm. between th |**Futures**:result of asyn. computation in thread pool `/*result in future obj*/FutureTask <LinearRegressionResult> future = new FutureTask<Integer>(new Callable <String>(){public Integer call(){ List<Doubles> lr = getDailyTurnover (..); return calcLinearRegression( lr);}; threadpool.execute(future);` |**SEDA**Staged event-driven architecture, a network of stages connected by event queues ++: Support massive concurrency, Simplify the construction of services(Provide abstractions), Enable introspection(Adapt behavior to changing load conditions), Support self-tuning resource managment, Server is created out of many SEDA stages |**Actor**:No access to shared memory(No locks,no deadlock), Scheduled on top of threads |**Reactive Programming**: oriented around data flows and propagation of change(Dependency graph) |||**App S**: a component-based product that resides in the middle-tier of a server-centric architecture. It provides middleware services for security and state maintenance, along with data access and persistence. Java:WildFly, Websphere |**EJB**: Standard component architecture, Distributed business applications(Support development, deployment and use of web services), Write once, run in all application containers ++:Developer not to care about Fail-over,Clustering,(Distributed) Transaction handling,Databases,Security,Deployment

10 Publish/Subscribe

Many-to-many: For communication, For coordination, M: data sources (publishing C) n: data sinks (subscribing C) C decoupled and do not know each other ↔ tightly coupled Observer Design Pattern: Removes explicit dependencies, Reduces coordination synchronization, Increases scalability of dis. SY, Creates highly dynamic(frequent add/remove) decentralized SY, Decoupling in three dimensions(Space Dec. No need for C (publishers & subscribers) to hold references or know each other, C physically distributed, Time Dec.C not to be available same time(Event in Buffer) Synchronization Dec.Control flow not blocked by the interaction)

Comparison: Decoupling in Different Interaction Schemes

Abstraction	Space decoupling	Time decoupling	Synchronization decoupling
Message Passing point to point	No	No	Producer-side
RPC/RMI Client Server	No	No	Producer-side
Asynchronous RPC/RMI	No	No	Yes
Future RPC/RMI	No	No	Yes
Notifications (Observer D. Pattern)	No	No	Yes
Tuple Spaces like event data base	Yes	Yes	Producer-side
Message Queuing (Pull)	Yes	Yes	Producer-side
Publish/Subscribe	Yes	Yes	Yes

Pub/Sub Models: Content-based, Channel-based(non-hierarchical), Topic-based(hierarchical, sport new USA) **Matching&Filtering in Content-based**: event(Publication) e, set of subscriptions S, find all subscription $s \in S$ matching e. Subscription: Boolean function over predicates Publication(event): Sets of attribute-value pairs **Two-phased Matching Algo.** 1.Match all predicates (Predicate Matching Phase) 2.Match subscriptions from results of Phase 1 (Subscriptions Matching Phase) **1.Predicate Mat.P.**: set P of predicates and event e, identify all satisfied predicates p of P → Predicate bit vector, Hash key = attribute name General Purpose Data Structure: for single attribute: 4 ordered linked list(b tree) for $=,<,>,!=$ operators, $O(n)$ all events & lists Finite Predicate Value Domain Types: huge matrix eg $1000 * 4\ forPrice \in [1,1000]$, 1000 for price, 4 for operators, entry lookup $m[i][j] = O(1)$ **2.Subscription Mat.P.: Counting Algo** |**Content-based Routing** 1.Advertisements (schema or types,data sources) Boradcast, 2.Publications & events (data sources)) 根据上一步Broker's Subscription走 , 3.Subscriptions (query, data sinks)

11 Service Orchestration

Business Process: set of linked activities which realise business goal Bus. Pro. vs Programs Granularity(Based on activities, Programming in the large), Control flow(Explicitly defined, Easy understand), Flexibility(change), Execution(call third-party webservices, Scheduled by process engine) VS Granularity(Based on instructions, Programming in the small) Control flow(Implicitly defined, Not easy understand) No Flexibility(Hard-coded) Execution(Invoke instructions locally, Scheduled by operating system) **BPEL, WS-BPEL**:Web Services Business Process Execution Language(XML based), to orchestrate loosely coupled services(to model SOA's Business processe), using web services running at bus. proc. execution engine (ODE), Lacks possibility for formal verification compared to FSP Components: Activities, State handling, Control structures, Exception handling, Partner links, Parallism, Dead path elimination Related Standards SOAP Simple Object Access Protocol, Comm. platform in XML WSDL Web Service Definition Language, set of endpoints operating on messages, Operations and messages described abstractly and bound to concrete network protocol UDDI Universal Description, Discovery and Integration, Services registered, published and reused by other organizations, App store for webservices

Service-Oriented ArchitectureSOA technique that involves the interaction between loosely coupled services that function independent Principles: Loose coupling, Service contract → WSDL, Abstraction of underlying logic, Autonomy, Reusability, Composability → WS-BPEL, Interoperability, Discoverability → UDDI |**BPMN vs BPEL** Business friendly, intuitive, process oriented, Swimlanes that represent organizational units, User friendly data manipulations, include human tasks, expose web service UI VS technical, manipulated with XPath expressions, xpress orchestrations, error handling, compensating actions |**Service composition** conceptual model: Servers exchanged Msg via out-port, in-port. design methodology: 1Bottom-up: Service providers develop and publish services Service consumers discover and select services Service consumers compose selected services 2Top-down: Service consumer develop a global process Service consumers decompose the global process into subprocesses Service consumers select/develop services to implement subprocesses |**Service orchestration** Local perspective, Describe control from one party's behavior(perspective), WS-BPEL as Standard, Executable processes which interact(message level) with web services (Service internal / external), Business processes(business logic + task execution order, span multiple apps and organizations, Define a long-lived transactional multistep process model) |**Service choreography** Global perspective, Describe the global interactions among all the parties, WS-CDL as Standard Tracks the message sequences among multiple source and sinks, Each involved party describes its part of the interaction |**Orchest. vs Choreo.** Two Orchest.(Web Service) sending each other(Choreo.) Request, ACK, Accept, ACK |**Formal Methods:** Modeling approaches, State machine verification, (Finite state processes (FSP), BPEL → LTS → FSP), to represent and reason about Complexity of Concurrency Sy, Performance optimization, Deadlock detection, dead path elimination |**FSP** Labeled Transition System(Abstract machine to study computation Contains a set of states and transitions between states) |**BPEL → FSP** Activitiy: <invoke partner='p1' operation='o1' /> → INVOKE = (invoke_p1_o1 -> END). <receive partner='p2' operation='o2' /> → RECEIVE = (receive_p2_o2 -> END). <reply partner='p1' operation='o1' /> → REPLY = (reply_p1_o1-> END). Sequence: <sequence> 见上面BPEL三个 </sequence> → 见上面FSP三个 SEQUENCE = INVOKE; RECEIVE; REPLY; END. $0 \to 1 \to 2 \to E$ Flow: Parallel Activity <flow> 见上面BPEL三个 </flow> → 见上面FSP三个 || FLOW = (INVOKE || RECEIVE || REPLY). $0 \to 1 \to 2 \to E \leftarrow 4 \leftarrow 5 \leftarrow 6 \leftarrow 7$ 而且每点出3种Activity |**Deadlock Detection**: when two or more competing activities are each waiting for the other to finish, in FSP: a non-final-state with no outgoing arcs, like A sends to B, B sends to A Assumption: Sync Comm $!m_1$: Send a Msg of type m, $?m_1$: Receive a Msg of type m