Simplifying Cyber Foraging

Rajesh Krishna Balan

Computer Science Department
Carnegie Mellon University

Motivation: mobile interactive applications

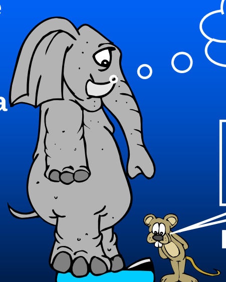
- speech recognition, language translation, augmented reality, …
  - Resource-heavy, but need bounded response time
    - Unfortunately, handhelds are weak!!

Columbia U. MARS project

2

Motivation: Handhelds are weak!

- Resource intensive App
- Huge Data Sets



2 GHz, 1 GB,
3-D graphics
2 GB of data

200 MHz, 32 MB,
no 3-D, no FPU
32 MB Flash

Resource-poor wearable

Poor performance!

3

Solution: Cyber Foraging

- "To live off the land"
  - Use resources in environment
- Remote execution
  - Augment device capabilities
- Fidelity Adaptation
  - Reduce resource usage
- OS Component of Aura

4

Problem #1

- How do we get large applications to work on small devices?
- Made more difficult by the mobile domain
  - Bandwidth fluctuates wildly
  - Battery concerns require variable operating modalities
    - Conserve battery at expense of quality and latency
    - Vice versa

5

Problem #2

- Seems likely that middleware can be built
  - With enough sweat, engineering, and plumbing
- But, what about the applications??

6

Problem : Retargeting Applications

- Large number of useful apps already exist
  - Complicated and large (> 100K lines of code)
  - Infeasible to write them from scratch
- Must be able to quickly retarget apps
  - Solution should be language independent
  - State of the art is 2 - 4 weeks per application

7

Thesis Statement

- It is possible to easily, quickly, and effectively modify an important class of existing computationally-intensive applications, such as language translators, speech recognizers, face detectors, and graphics applications, for cyber foraging.

8

Strawman Solutions

- Run all applications on remote server
  - No application modifications needed
  - Use ssh or vnc to access server
  - Network latency is a problem
  - Requires bandwidth
  - No degraded / corrective failure mode
- Run all applications locally
  - Not enough resources

9

Roadmap

- Motivation & Background
- Solution Requirements
- Systems + Software Engineering Solution
- Evaluation
- Conclusion

10

Requirement No. 1

- Support as many applications as possible
  - No restriction on language
    - Must support C, C++, Java, .NET, etc.
  - No restriction on programming style
    - Procedural, functional, event-based (OpenGL), etc.
  - No restriction on type of application
    - Language translators, face detectors, graphics apps

11

Requirement No. 2

- Target Entry-level Application Developers
  - Effectively fresh hires right out of college
    - Representative of who actually does this work
  - Assume no expert knowledge of anything
    - Basic programming skills assumed

12

Requirement No. 3

- Solution must be good
  - Retargeting time must be low
    - < 6 hrs per application
    - Current time is ~ 2 to 4 weeks per application
  - Retargeted application is isolated
    - From runtime / OS / hardware changes
    - Vital for fast moving mobile market
  - Performance must be excellent
    - Comparable to hand-retargeted applications

13

Summary of Requirements

- Any application
- Any developer
- Fast
- Good

➔ Sounds Impossible

14

Key Insight

Application aspects relevant for remote execution and fidelity adaptation can be expressed in a short declarative form

- Expected Application Resource Usage
  - Use concept of *parameters*
- Runtime adaptive aspects of application
  - Application runtime settings => *fidelity variables*
  - Remote execution partitionings => *tactics*

15

Roadmap

- Motivation & Background
- Solution Requirements
- Systems + Software Engineering Solution
- Evaluation
- Conclusion

16

Application Domain

Applications must satisfy three criteria

1. Useful Mobile Application
   - Language translation, speech recognition etc.
2. Resource Constrained wrt. Mobile Devices
   - Eliminates email, calendar etc.
3. Interactive
   - User provides input, application does work. Repeat
   - Eliminates media streaming type applications

17

Design Considerations

- Achieve excellent mobile performance
  - Make dynamic decisions
  - Requires resource measurements, resource predictions, "optimal" solver

  ➔ Large amount of extra code needed for each application
- Keep retargeting cost low
  - Minimize application modifications
  - Abstract as much as possible
  - Place common functionality in common layers

18

Solution Design

- Coarse-grained remote execution
  – Modular level (RPCs)
  – Finer grain requires language support
- Cannot wrap around existing APIs
  – Application / language independent
  ➔ Applications must be hand-modified
    - Cannot use generic code analysis techniques

19

Design Considerations

- Achieve excellent mobile performance
  – Make dynamic decisions
  – Requires resource measurements, resource predictions, “optimal” solver
  ➔ Large amount of extra code needed for each application
- Keep retargeting cost low
  – Minimize application modifications
  – Abstract as much as possible
  – Place common functionality in common layers

20

Components of Solution

1. Language for describing applications
   - Vivendi
2. Adaptive Runtime System
   - Chroma
   - Handles common adaptation requirements
3. Smart Stub Generator
   - Generates most of the app to runtime interface code
4. Well-defined procedure for modifying app

21

Vivendi

APPLICATION pangloss-lite;

IN int num_words;
OUT float quality;

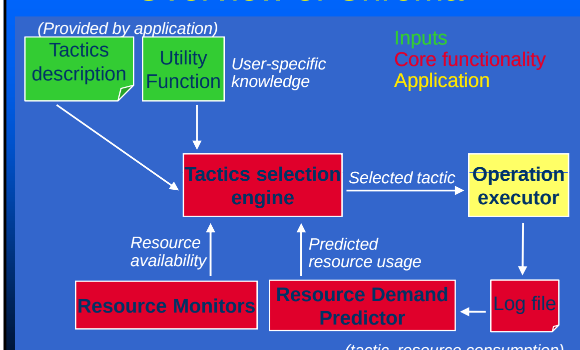
Dictionary → Language modeler
Example based → Language modeler

/* RPC Specifications */
RPC server_dict (IN string line, OUT string dict_out);
RPC server_ebmt (IN string line, OUT string ebmt_out);
RPC server_lm (IN string gloss_out, IN string dict_out, IN string ebmt_out, OUT string translation);

/* Tactics (Useful Ways to Combine the RPCs) */
TACTIC dict = server_dict & server_lm;
TACTIC ebmt = server_ebmt & server_lm;
TACTIC dict_ebmt = (server_dict, server_ebmt) & server_lm;

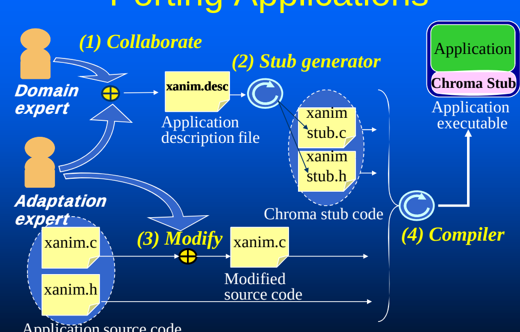
22

Overview of Chroma



23

Porting Applications



24

Client API Calls

Basic

- Register :- Register app with runtime
- Cleanup :- Remove app from runtime

Core Functionality

- Find_fidelity :- Asks runtime to decide appropriate runtime settings
- Do_tactics :- Perform the operation

25

Roadmap

- Motivation & Background
- Solution Requirements
- Systems + Software Engineering Solution
- Evaluation
  - Chroma
  - RapidRe
- Conclusion

26

Chroma: Evaluation Objective

- To show that Chroma has comparable performance to an oracle
  - Oracle selects best tactic for current environment
  - Oracle's selection is determined offline while Chroma selects online
- Is the overhead of Chroma acceptable?

27

Applications Used

- Three real research applications
  - Useful for mobile users

1. Pangloss-Lite is a natural language translator (7 tactics)
   - Valuable for travelers in foreign countries
2. Janus is a speech recognizer (2 tactics)
   - Key component of speech interfaces
3. Face detects faces in images (1 tactic)
   - Representative of surveillance applications

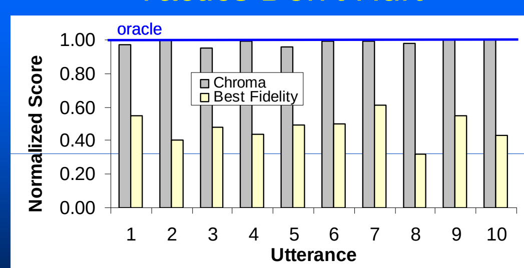
28

Experimental Setup

- Thinkpad 560X Client (200 Mhz Pentium)
  - Representative of fastest handhelds
- HP Omnibook 6000 Servers (1 Ghz Pentium 3)
  - unloaded except where stated otherwise
- 100 Mb/s Ethernet
- Testing methodology
  - Different inputs representing an operation
  - Each result average of 20 runs
  - State of system reset before each run

29

Tactics Don't Hurt



Application: Janus, Metric = $\frac{fidelity}{latency}$

30

Tactics Don't Hurt

Utterance	Oracle		Chroma		Score
	Latency (s)	Fidelity	Latency (s)	Fidelity	
1	0.71	0.50	0.73	0.50	0.97
2	1.00	0.50	1.00	0.50	1.00
3	0.76	0.50	0.80	0.50	0.95
4	0.78	0.50	0.79	0.50	0.99
5	0.99	0.50	1.03	0.50	0.96
6	0.95	0.50	0.96	0.50	0.99
7	0.70	0.50	0.71	0.50	0.99
8	1.20	0.50	1.22	0.50	0.98
9	0.77	0.50	0.77	0.50	1.00
10	0.99	0.50	0.99	0.50	1.00

Application: Janus, Metric = $\frac{\textit{fidelity}}{\textit{latency}}$

Location : Remote in all case

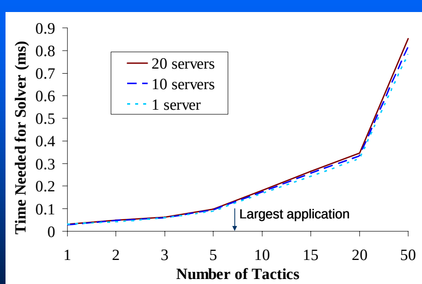
31

Overhead Concerns

- Does the solver take too long?
  - Can it handle a large number of tactics?
  - What happens when the number of servers increases?
- Complete system overhead
  - How long does the system need to make decisions on a slow client?
    - Includes solver overhead + overhead of resource measurement & prediction

32

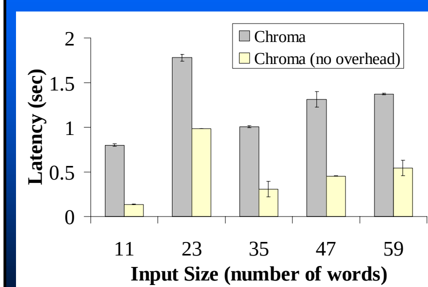
Overhead of Solver



- Synthetic results
- Slow client
- Max overhead < 0.9 ms
- Okay for interactive apps

33

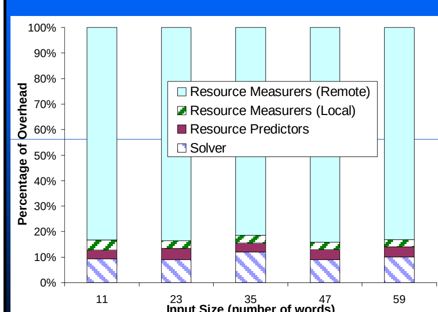
Overhead of Entire System



- Pangloss – Lite
- Slow client
- Overhead is high
  - Between 0.5 – 0.8s

34

Breaking Down the Overhead



- Measurers are slow
  - Caching helps
  - Tradeoff accuracy for latency

35

Chroma: Summary

- Achieves close to optimal performance
  - Errors due to bad resource measurements or predictions
- Overhead is reasonable
  - Brute force computation is okay
  - Resource values should be cached for optimal performance

36

Other Results In Thesis

- Overprovisioned Environments
  - Chroma can automatically use extra servers to improve application performance
- Integration with dynamic user preferences (Prism)
- Security
  - Simple mechanisms to detect rogue servers
- Fairness (In Cheng's thesis)

37

Roadmap

- Motivation & Background
- Solution Requirements
- Systems + Software Engineering Solution
- Evaluation
  - Chroma
  - RapidRe
- Conclusion

38

RapidRe: Evaluation Objectives

To show that RapidRe is:

- Quick
  - Retargeting time is low. < 6 hrs per app
- Easy
  - Users don't find the retargeting task difficult
- Effective
  - Retargeted apps equivalent to hand modified versions

39

Evaluation Methodology

- Conduct rigorous software usability study
  - Large number of real apps
  - Large number of representative developers
    - 13 Senior undergrads
    - Measure time taken
    - Measure number of errors in solutions
- Conduct systems evaluation
  - Compare user study apps with expert apps
    - Expert apps are hand retargeted for the same runtime

40

Test Applications

App	Lines of Code	File Count	Language	Functionality
Face	20K	105	Ada / C	Face Recognizer
Flite	570K	182	C	Speech Synthesis
GLVU	25K	155	C++ / OpenGL	3D Model Viewer
GOCR	30K	71	C++	Char Recognition
Janus	126K	227	C / Tcl / Motif	Speech Recognizer
Music	9K	55	Java / C++	Music Recognizer
Panlite	150K	349	C++	Language Translator
Radiator	65K	213	C++ / OpenGL	3D Lighting Models

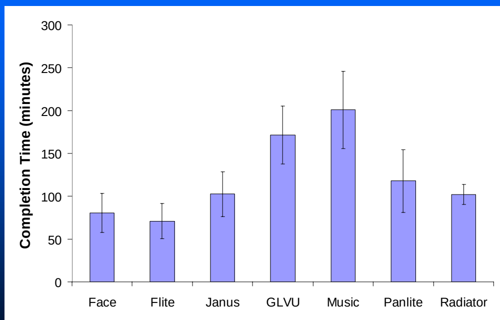
41

Procedure

- Train each participant for 1 hr
  - Using GOCR
- Measure the time needed for them to perform each stage of the process
  - Stage 1 : create tactics file
  - Stage 2 : create client component
  - Stage 3 : create server component

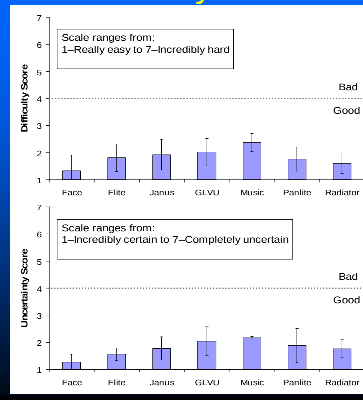
42

Modification Times



43

Difficulty and Confidence



Difficulty against Time
Correlation coeff = 0.82
Probability = 0. 02

Certainty against Time
Correlation coeff = 0.88
Probability = 0.009

Certainty and Difficulty
Correlation coeff = 0.88
Probability = 0.01

44

RapidRe: Summary 1

- Reduced modification times from weeks to under 4 hrs
- Very few errors in solutions (not shown)
- But what about performance of modified apps?
  - Test modified apps against expert created apps
    - Under various mobile computing scenarios (not listed)

45

Performance of Modified Apps

- Tactics files had few errors (not shown)
- Client App performance shown

	Participant Number												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Face	100%			100%							67%		
Flite								100%			100%	67%	
GLVU	44%		44%		44%		100%		44%				
Janus		100%	100%				100%						
Music			100%		100%								100%
Panlite		100%		83%		100%		100%		100%			
Radiator				94%						100%		78%	

- All server apps had equivalent performance

46

RapidRe: Summary 2

- Performance is also good
  - 25 / 25 server apps correct
  - 16 / 25 client apps correct
    - Not perfect but this was expected
- Performance problems due to two factors
  - Failure to specify application resource usage
  - Failure to use returned fidelity values
    - Measures can be taken to reduce these errors

47

Other Results In Thesis

- Detailed Breakdown of Errors
- Additional User-Perceived Results
  - Amount of Isolation
  - Effect of Experience
- Detailed Process Analysis and Suggestions for Improvements

48

Roadmap

- Motivation & Background
- Solution Requirements
- Systems + Software Engineering Solution
- Evaluation
- Conclusion

Key Challenges and Solution

- Any language
  - Runtime written in C
- Any developer
  - Vivendi + well-defined API insertion rules + proper isolation + good documentation
- Fast
  - Proper abstraction + stub generator + good documentation
- Easy
  - Proper abstraction + isolation + good documentation
- Effective
  - Chroma + Prism Integration

Thesis Requirements Summary

- Any application ➔ Yes!
- Any developer ➔ Yes!
- Fast ➔ Yes!
- Good ➔ Mostly (can be Yes!)

➔ There must be a catch

Reasons for Success

- Exploiting previously unknown similarities common to all the applications
  - Commonly used software development methods encourage proper modularization
- Naturally causes the ``thin waistband'' needed for this method
  - Facilitates high level description
  - Makes it easy for novices to add APIs
    - Interactive app model facilitates this as well

Only Part of the Solution

- 4 big missing pieces still needed
  - Integrating user preferences
    - Work by Sousa and Poladian (Aura project)
  - Application hint modules
    - Map fidelities / resources to application quality
  - Create appropriate UIs for a mobile device
    - Work by Eisenstein et al. and Myers et al.
  - Actual test deployment

Some Related Work

- Dynamic runtime systems
  - Odyssey [Noble], Puppeteer [deLara], Rover [Joseph], Coign [Hunt]
- Little Languages
  - Make [Feldman], QML [Frolund], QDL [Loyal], Aspect Languages [Kiczales]
- Retargeting Frameworks
  - IBM's WebSphere, Microsoft's .NET Framework

Summary

- Big challenge in modern computing
  - Design high performance systems that are provably usable
- Systems are usually fast enough
- However, they are frequently not usable
- This is a concerted attempt to bridge the gap
  - Using techniques from very diverse areas

55

The End

- Questions?

56

Benefits of Solution

- Developer describes app using Vivendi
  - Knowledge of small part of app needed
- Stub creates code to interface with Chroma
  - Knowledge of Chroma unnecessary
  - Easy well defined process to insert APIs into app
- App APIs are runtime / OS agnostic
  - Only stub generator needs to be modified
    - Apps just need to be recompiled with new stub code
    - Similar to glibc changes

57

Didn't Talk About

- Chroma Runtime System in detail
- Other research
  - Distributed infrastructure for supporting massively multiplayer games
  - Virtual machine techniques to improve scientific applications
  - Changes to TCP's protocol
    - Improve wireless network performance
    - Reduce power consumption

58

App-Inserted API Calls (Server)

Basic

- Service_init :- Register app with runtime

Core Functionality

- Run_server :- Start the stub-generated event loop

Other Modifications

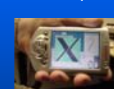
- Create RPC stubs required by server
  - The RPCs specified in the vivendi syntax file

59

Adjusting to Environment

- Availability of compute servers varies wildly
  - Mobility ⇒ Cannot expect just one situation

Resource Poor



Resource Rich (Smart Rooms etc.)

- Tactics allow us to automatically use extra resources in environment
  - Without modifying the application

60

Why is this Useful?

- Shields application from uncertainty in environment
  - Load on servers, wireless bandwidth
  - Execute same tactic on multiple servers
    - Take fastest result
- Opportunistic execution
  - To meet latency constraints
  - Execute multiple tactics on multiple servers
    - Return highest fidelity result that satisfies latency constraints

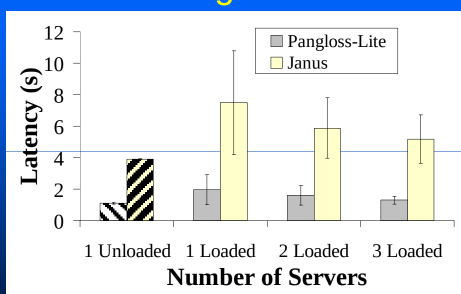
61

Additional Benefits

- Code is easier to test and debug
  - Very few developer-added lines
  - Most of the new code is stub generated
- Experts can now do the retargeting
  - Should require less time than novices
  - Resulting applications should be excellent

62

Tactics: Using Extra Servers



Still able to get performance improvements with loaded servers!!

63

Meeting Latency Constraints

	Fidelity	Latency		Metric
		Average (s)	Standard Deviation (s)	
Running to Completion	1.0	1.96	0.15	0.51
Taking Best Result after 1s	0.75	1.00	0.01	0.75

Application = Pangloss-Lite

Thinkpad 560X (200 Mhz Pentium)

$$\text{Metric} = \frac{fidelity}{latency}$$

64

Two–Phase Evaluation

- Chroma is good
  - Achieves near-optimal performance in mobile environments
  - Overhead is low
- RapidRe is good
  - Allows legacy applications to be easily, quickly, and effectively retargeted to use Chroma

65

Applications

App	Lines of Code	File Count	Tactic File Size	Stage B: Client Modifications				Stage C: Server Modifications			
				Lines Added	Lines Removed	Stub Lines	Files Changed	Lines Added	Lines Removed	Stub Lines	Files Changed
Face	20K	105	10	31 – 68	12 – 15	556	2	26 – 45	15 – 21	186	2
Flite	570K	182	10	29 – 39	1 – 5	556	2	13 – 30	3 – 87	186	2
GLVU	25K	155	38	62 – 114	3 – 21	1146	2	88 – 148	12 – 32	324	2
Janus	126K	227	25	28 – 47	2 – 7	1538	3	59 – 130	7 – 70	434	4
Music	9K	55	11	61 – 77	4 – 6	1127	2	131 – 209	23 – 147	203	2
Panlite	150K	349	21	30 – 66	1 – 39	1481	3	12 – 73	18 – 39	406	3
Radiator	65K	213	15	41 – 51	1 – 47	643	2	49 – 106	17 – 32	202	2

66

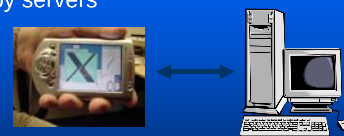
Application Modification Times

App	Stage A	Stage B	Stage C	Total
Face	10.3 (1.7)	36.6 (4.5)	33.6 (17.8)	80.5 (22.7)
Flite	12.6 (7.8)	37.7 (6.7)	20.6 (16.4)	70.9 (20.4)
Janus	29.3 (14.0)	31.0 (6.5)	42.1 (10.2)	102.4 (26.2)
GLVU	66.3 (20.8)	65.1 (22.5)	40.3 (7.7)	171.7 (33.8)
Music	49.6 (15.7)	68.2 (17.1)	83.0 (23.0)	200.8 (45.4)
Panlite	36.2 (7.7)	48.7 (20.2)	32.8 (14.7)	117.8 (36.6)
Radiator	17.2 (6.0)	45.3 (8.7)	39.4 (7.0)	101.9 (11.7)

67

Solution: Remote Execution

- Augment capabilities of handhelds by using nearby servers



- But how can good performance be achieved in mobile environments?
- And easily allow legacy applications to use remote execution?

68

Performance of Modified Apps

- Tactics files had few errors (not shown)
- Client App performance shown

Participant Number

	1	2	3	4	5	6	7	8	9	10	11	12	13
Face	100%			100%							57%		
Flite								100%			100%	67%	
GLVU	44%		44%		44%		100%		44%				
Janus		100%	100%				100%						
Music			100%		100%								100%
Panlite		100%		83%		100%		100%		100%			
Radiator				94%						100%		78%	

- All server apps had equivalent performance

69

Key Insight

For a large number of applications

- Number of useful remote partitions is small
  - Largest so far is 7 partitions
    - Modular level coarse-grained partitions
- Application developer specifies these partitions (static partitioning)
  - At runtime, pick the optimal partition and locations (dynamic partitioning)

70

Performance of Modified Apps

- Tactics files had few errors (not shown)
- Client App performance shown

Participant Number

	1	2	3	4	5	6	7	8	9	10	11	12	13
Face	100%			100%							57%		
Flite								100%			100%	67%	
GLVU	44%		44%		44%		100%		44%				
Janus		100%	100%				100%						
Music			100%		100%								100%
Panlite		100%		83%		100%		100%		100%			
Radiator				94%						100%		78%	

- All server apps had equivalent performance

71

Solution: Tactics

- Concise description of application's remote execution capabilities
  - Only the useful remote partitions are described
  - Can be captured in a compact declarative form
  - Each tactic performs the required operation
    - Operation ➔ application specific notion of work
- Tradeoff between dynamic and static partitioning
  - RPC model
  - Assume servers have been discovered and are able to handle any RPC call (no code migration)
  - Coarse-grained remote execution

72

Too good to be true?

- Results are incredibly good
  - < 3 ½ hours for novices to retarget complex applications with good performance
- What's the Catch?

73

Validation of Ease of Use

- Conduct Software Usability Study
  - Using senior undergrads
- Modify a large number of real applications
  - 7 applications + 1 training application
  - Created by other groups
  - Written in many languages
    - Java, C, C++, Ada, TCL/TK

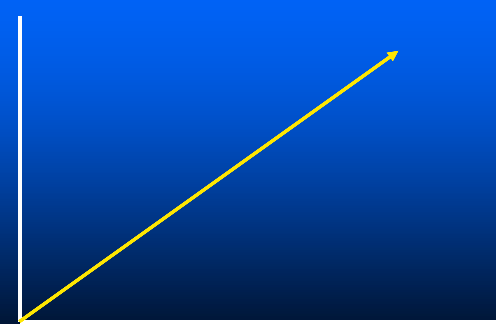
74

Background

- Proliferation of Mobile Devices
  - Cellphones, PDAs, watches
  - Laptops are not really mobile devices
- However, smaller size ➔ compromise CPU and battery
  - Unable to run large applications
- But, these applications are highly useful
  - Language translation
  - Speech recognition
  - Pattern recognition (signs, text etc.)

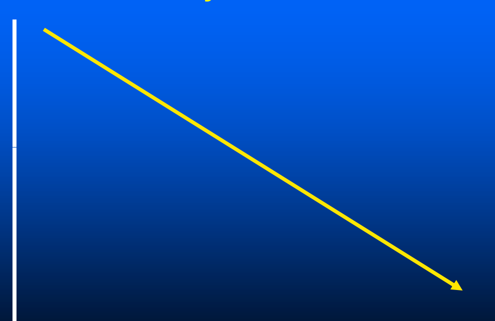
75

Satya Slide



76

Satya Slide

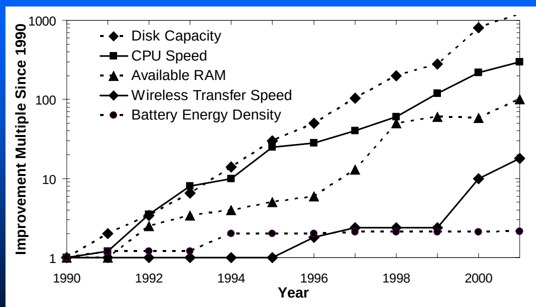


77

Satya Slide

78

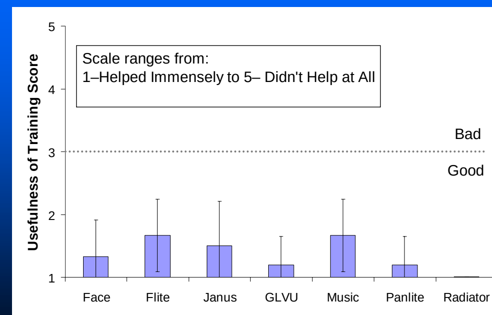
Moore's Law



Courtesy Thad Starner, Georgia Tech

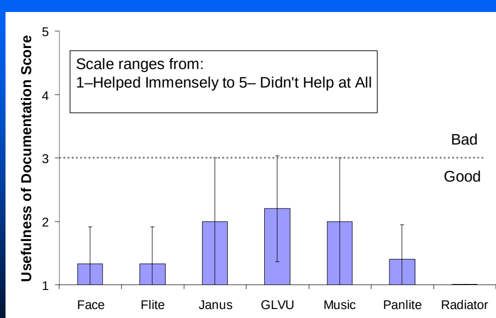
79

Usefulness of Training



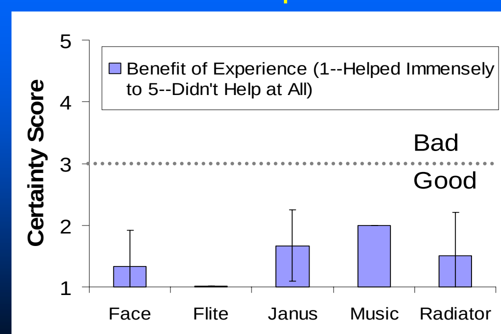
80

Usefulness of Documentation



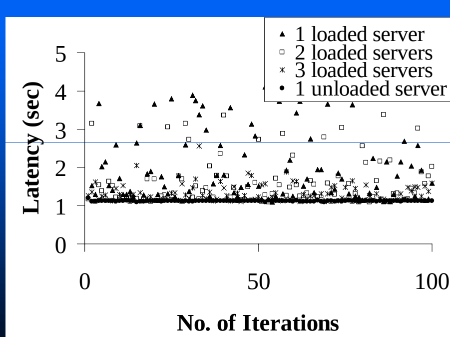
81

Effect of Experience



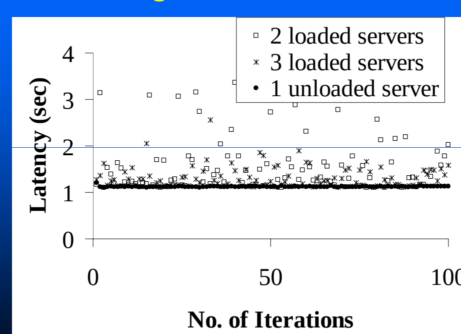
82

Using Loaded Servers



83

Using Loaded Servers



84

Stages for User Study

Stage A *Tactics File*	Stage B *Client Component*	Stage C *Server Component*
Read Docs	Read Docs	Read Docs
Application	Include file	Include file header
In	Register	*service_init* API call
Out	Cleanup	Create RPCs
RPC	Find Fidelities	*run_server* API call
Tactic	Do Tactics	Compile and fix
	Compile and fix	

Errors in Tactics File

Apps	Params	RPCs	Tactics	Harmless	# Apps	Okay
Face	0	0	0	0	3	3
Flite	1	0	0	0	3	2
GLVU	1	1	0	3	5	4
Janus	0	0	0	1	3	3
Music	0	1	0	0	3	2
Panlite	0	0	2	0	5	3
Radiator	0	2	0	0	3	1
Total	2	2	0	4	25	18

Errors in Stages B and C

	Trival Errors					Non-Trivial Errors	
Apps	Reg. Late	Output File	Output Space	Mem. Freed	Other	Params	Fids
Face	0	3	0	0	0	1	0
Flite	0	3	0	0	1	0	0
GLVU	3	0	1	0	1	1	4
Janus	1	2	0	0	0	0	0
Music	1	0	0	2	0	1	0
Panlite	4	0	0	0	0	1	0
Radiator	2	1	0	0	0	2	0
Total	11	9	1	2	2	6	4

Experimental Scenarios

ID	Load	BW	User Prefs
Q	Low	High	Highest quality result
T	Low	High	Lowest latency result
LH	Low	High	Highest quality result within X seconds
HH	High	High	Highest quality result within X seconds
LL	Low	Low	Highest quality result within X seconds
HL	High	Low	Highest quality result within X seconds