Hang Analysis: Fighting Responsiveness Bugs

{Xi Wang, Zhilei Xu}[1] {Zhenyu Guo, Xuezheng Liu,Haoxiang Lin, Xiaoge Wang, Zheng Zhang} [2]

[1]Tsinghua University [2]Microsoft Research Asia

Présentée par : Quoc Anh LE

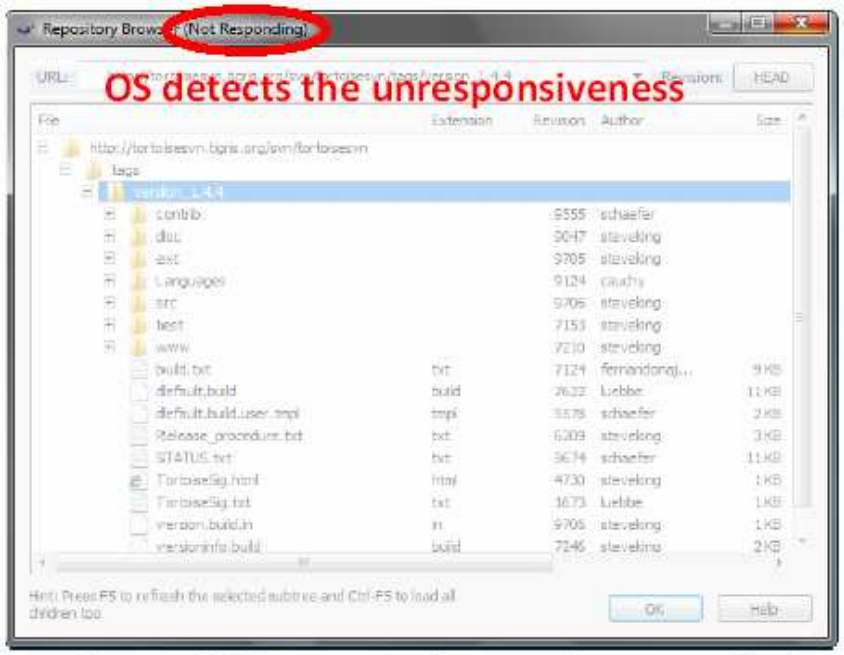
Cours: Système d'exploitation M.A / INFO M467

Professeur: Monsieur Jean RAMAEKERS

FUNDP NAMUR

Qu'est-ce que c'est « soft hang »?

- L'ordinateur est dans le coma au lieu de devoir réponse à une action de l'utilisateur sur le coup
- L'utilisateur ne peut pas faire l'autre requête pendant le « soft hang »
- Cause: Blocage ou Erreur de timeout => à partir des « hang points »



(a) TortoiseSVN repository browser (not responding).

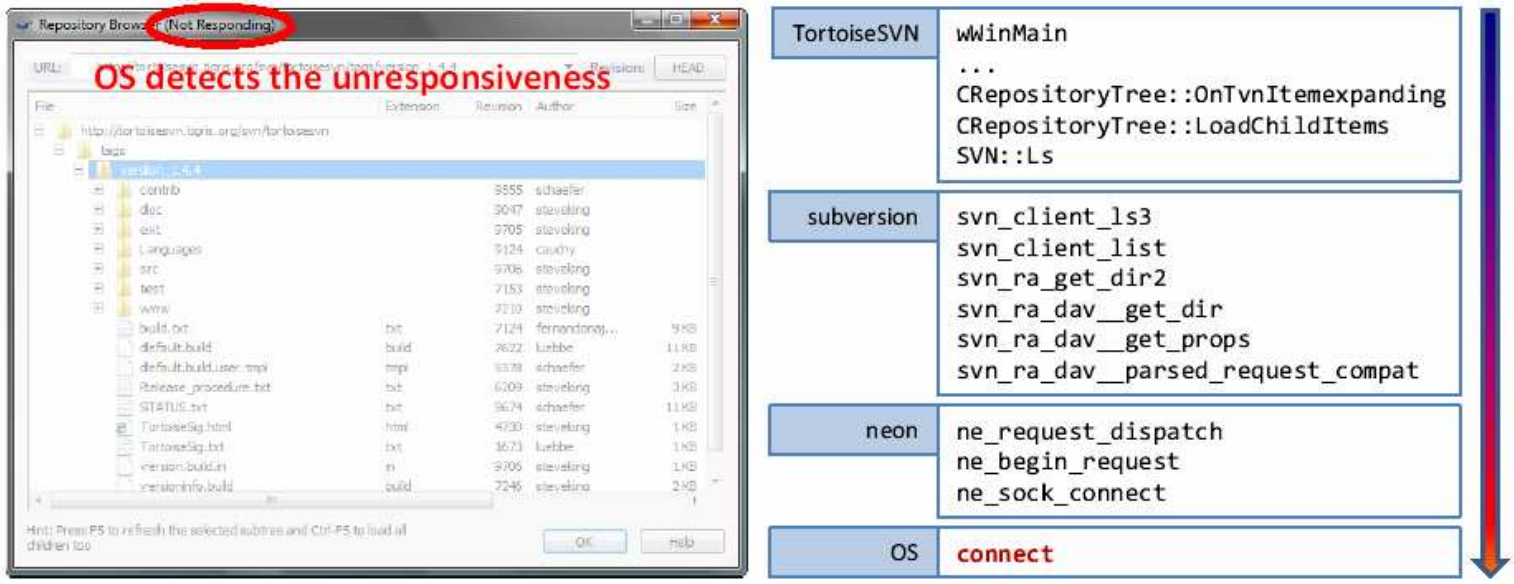
Qu'est-ce que c'est un « hang point »?

- Une invocation est en mesure de compléter rapidement tel qu'une action sur l'interface graphique mais elle appelle une fonction de blocage.
- L'application revivra éventuellement mais pendant un longtemps, l'utilisateur ne peut ni annuler ni fermer l'application.
- Il peut être un deadlock, un boucle infini ou une erreur concernant l'interruption

L'objectif du travail

- Collecter les patterns des « hang points »
- Chercher exhaustivement les « hang points » en comparant avec les patterns pour la correction.

Exemple d'un analyse de Tortoise



(a) TortoiseSVN repository browser (not responding). (b) A snapshot of the UI thread's call stack.

Figure 1: A simple hang case of TortoiseSVN 1.4 repository browser. When a user clicks in the tree view to expand a directory node of a remote source repository in the repository browser, the UI thread calls back to the event handler function `CRepositoryTree::OnTvnItemexpanding` in TortoiseSVN, which eventually invokes a blocking socket API function `connect`. Thus, its UI stops responding and the application hangs during network communication.

Cause: L'erreur de « connect » + l'erreur de timeout

Solution:

- devoir être un thread séparé
- pouvoir être optionnel annulé

Difficulté de collection des patterns

- « hang » d'une de la chaîne des opérateurs
- Tierce module ou Plug-ins
- Manque des outils effectifs
  - Ne pas pouvoir couvrir tous les cas
  - Sensible à l'environnement

Collection des patterns responsives

PATTERN 2.1. *An invocation to any function in the UI thread is responsive.*

- *Responsive invocations* sont les invocations souhaitées de finir sur le coup
- Un programme ne peut que continuer après la finition de ses *responsives invocations* aux autres fonctions.
- Si une fonction est *responsive*, alors toutes invocations de cette fonction à l'autre fonction sont *responsives*
- Par exemple la fonction CRespositoryTree:OnTvnItemexpanding est *responsive*, alors Connect est aussi *responsive*

Collection des patterns blockings

PATTERN 2.2. *An invocation to* `connect` *is blocking.*

PATTERN 2.3. *An invocation to* `GetFileAttributesW` *is blocking if the file path is may-remote.*

- *Blocking Invocations* sont peut-être des fonctions « wait » ou « sleep », communication, … c'est-à-dire les invocations qui consomment un temps
- Le pattern 2 ne dépend pas des paramètres ou du contexte appelé. Donc elle est appelée « **inconditionnel pattern** »
- Le pattern 3 dépend des paramètres et du contexte appelé, donc elle est appelée « **conditionnel pattern** »

Exemples des patterns conditionnels blocking

property	example	brief descrption	blocking condition
may-remote	`GetFileAttributesW(` `lpFileName)`	get file information	`lpFileName` is remote
may-nonzero	`Sleep(` `dwMilliseconds)`	sleep for an interval	`dwMilliseconds` is nonzero
may-null	`TransmitFile(` `...,lpOverlapped,...)`	send file data over a socket asynchronously, or synchronously if `lpOverlapped` is null	`lpOverlapped` is null

Table 1: Examples of conditional blocking patterns. `GetFileAttributesW`, `Sleep` and `TransmitFile` are Win32 API counterparts to POSIX API `stat`, `sleep` and `sendfile`, respectively.

par exemple, Sleep(0) ne fait pas un blocage mais Sleep(50) fait un blocage

Hang Bugs

- *Hang bugs* est une invocation qui **est *responsive* et aussi *blocking***
- Ex: Pour le cas TortoisSVN, selon le Pattern 1, la méthode OnTvnItemexpanding est *responsive*, donc *connect* est *responsive*. Selon Pattern 2, *connect* est *blocking*. Donc, c'est un *hang*.
- De façon de chercher exhaustivement les codes sources en utilisant les patterns, on peut identifier les *hang*s.

Architecture général du système

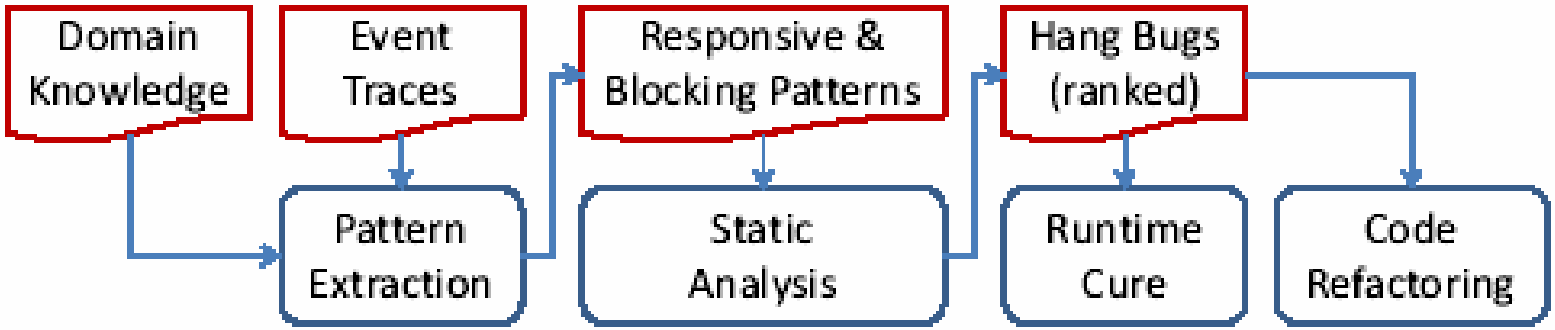


Figure 3: Architecture of the hang analysis system.

- *Pattern extraction:* collecter les patterns crucials par les experts ou les tests
- *Static Analysis*: Patterns responsives + patterns blockings + codes sources = hang bugs
- *Cure and refactoring*: suggestion pour fixer les *hangs* détectés

Algorithme d'analyse « hang »

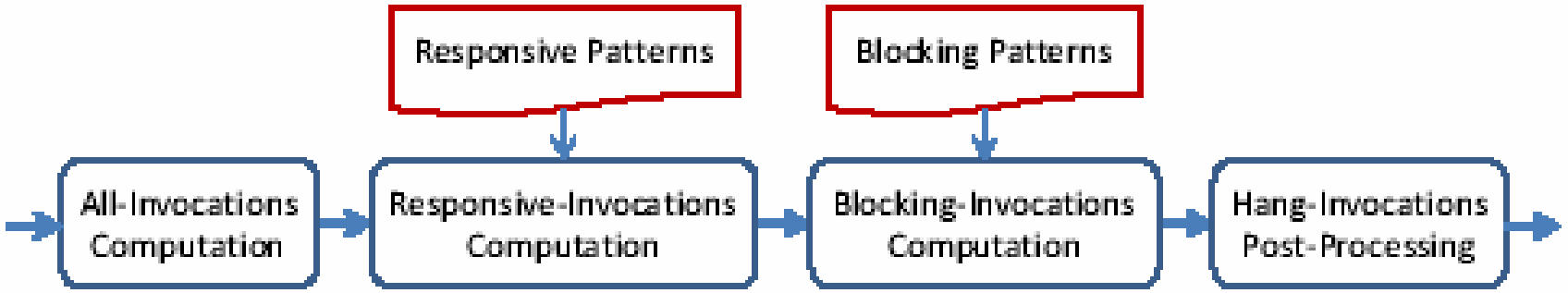
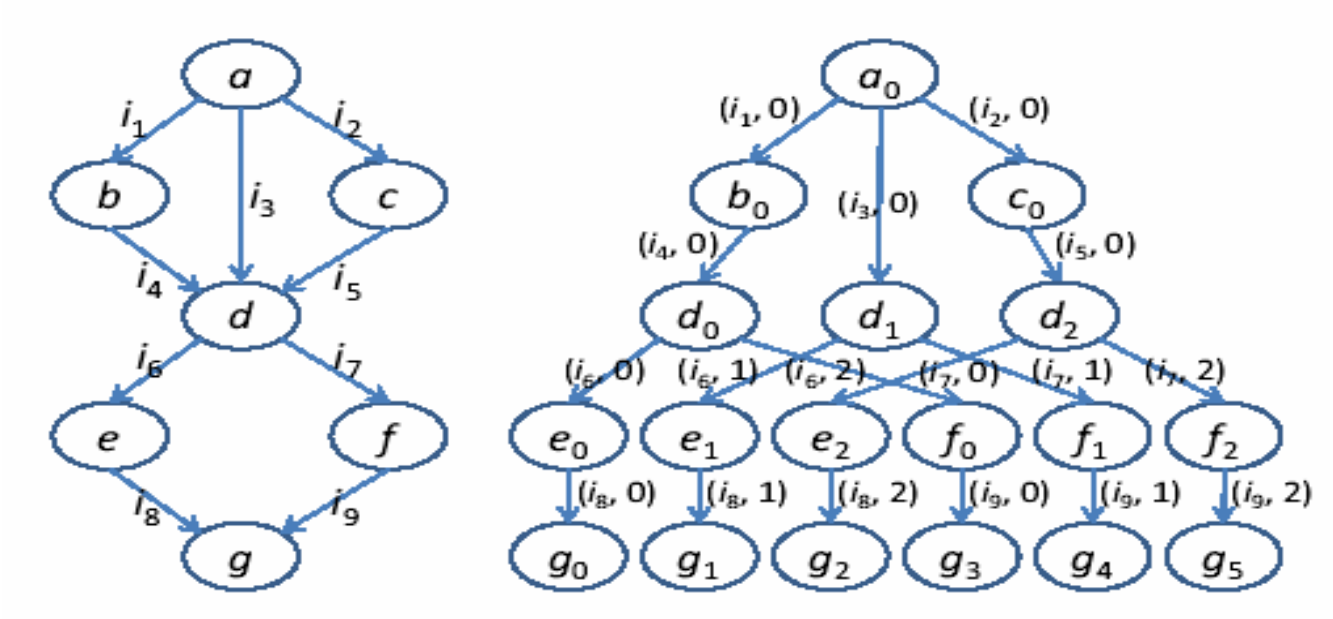


Figure 4: Stages of the hang analysis algorithm.

$$Reach = \{f | (\texttt{main}, f) \in call\} \cup \{f | \exists g \in Reach : (g, f) \in call\}$$

- Définir une relation *call*(fonction appeleur, fonction appelée)
- A partir de la fonction *main*, on arriver tous les fonctions possibles dans les codes sources

Détection de toutes invocations



- Invocation conditionnelle dépend des paramètres et du contexte. Donc, on doit préciser les propriétés sur les chemins individuels comme le figure à droit ci-dessus.
- Chaque côté représente une invocation (i,c) dans laquelle i est la fonction appeleur et c est un contexte

Détections des invocations responsives

$$R_k = ResonsiveRule_k(AllIvk)$$

$$ResponsiveIvk_0 = \bigcup R_k$$

$$ResponsiveIvk = \{(i,c)|(i,c) \in AllIvk$$
$$\wedge \exists (i',c') \in ResponsiveIvk_0 : (i',c') \longrightarrow^* (i,c)\}$$

- A partir d'un ensemble d'invocations responsives (obtenu en comparant toutes les invocations détectées avec le Pattern 2.1)
- Grâce à la caractéristique «Si une fonction est *responsive*, alors toutes invocations de cette fonction à l'autre fonction sont *responsives »*

Détections des invocations blockings

$$B_k = BlockingRule_k(AllIvk)$$

$$BlockingIvk = \bigcup B_k$$

- A partir des patterns, on déduit un ensemble initial des invocations blocking

$$B_1 = \{(i,c) | (i,c) \in AllIvk \wedge \mathcal{F}(i) = \texttt{connect}\}$$

$$B_2 = \{(i,c) | (i,c) \in AllIvk \wedge \mathcal{F}(i) = \texttt{GetFileAttributesW} \wedge (\mathcal{P}(i,1),c) \in may\text{-}remote\}$$

- F(i) est fonction appelée, P(i,1) retourne la première variable de la fonction i.
- *May-remote* est calculé en vérifiant le string comme le chemin

Calcul des Hang-Invocations

- Hang invocation = ResponsiveIvk $\wedge$ BlockingIvk
- Classifier les hangs invocations:
  - *Hot call site*: Si un site cause « hang » dans plusieurs contextes, il est plus haut de la classification.
  - *Hot call path*: Certaines fonctions sont utilisées plus que les autres donc, leurs invocations sont plus hautes de la classification
- Priorité des corrections des hang invocations est basée sur leur classification

Difficultés

- La collection des patterns n'est pas complète parce que
  - Pour les plug-ins, les codes sources de l'appelé est manqués
  - Les libraires externes, chargé dynamique,..
  - Les évaluations de la condition de blocking dépendent les paramètres et du contexte

Implémentation

- Utiliser Phoenix pour analyser les sources de C/C++ et C#/.NET (Phoenix, c'est un logiciel de Microsoft)
- Patterns responsives sont fournis par les experts
- Patterns blocking sont collectés à partir deux sources:
  - Fournis par l'équipe des développeurs + API Documents
  - Suivants manuellement sur les applications de bureau

Evaluation

- Les auteurs ont créé un Système HANGWIZ
- Windows Server 2003 x64 avec Intel Xeon 2.0 Ghz CPU, 32 GB de mémoire
- 102 patterns blocking (53 inconditionnels et 49 conditionnels)
- Résultats:

	analysis time	reported invocations	false alarms	hangs	
				invocations	gates
TortoiseSVN	17m30s	229	77	152	26
GIMP	1h29m57s	69	35	34	10
lighttpd	3m19s	33	9	24	10
Total	–	331	121	210	46

Table 3: Experimental results.

Élimination des « hang points »

org.eclipse.update.internal.core.SiteFileFactory

```
private void parsePackagedPlugins(File pluginDir) … {
                                        ...
    String[] dir = pluginDir.list(…);
                                        ...
    for (int i = 0; i < dir.length; i++) {
                                        ...
        File file = new File(pluginDir, dir[i]);
        jarReference = new JarContentReference(null, file);
        ref = jarReference.peek("META-INF/MANIFEST.MF", …);
        if (ref != null) {
            in = ref.getInputStream();
```

Figure 6: Example of long-running loops in Eclipse 3.3. The stop condition depends on `dir.Length` of the outer environment, and thus the loop may take a long time.

- Générer un thread séparé pour chaque tâche
- Une option permettant à l'utilisateur d'annuler la tâche
- Etablir les drapeaux pour limiter le temps d'exécution

Conclusion

- L'article présente un modèle simple et effectif
- *Hang bugs* n'est pas *correctness bugs* (Une instance typique de *correctness bugs* est *deadlock*). Donc, le modèle proposé est particulier.
- Les techniques d'apprentissage sont peut-être appliqués pour extraire les patterns
- Il reste encore certaines difficultés