**Site outage 2/16/2018 Sitecore response**

There are a lot of locks with such stacktraces:

System.Text.RegularExpressions.Regex.LookupCachedAndUpdate(System.String)+5a
System.Text.RegularExpressions.Regex..ctor(System.String, System.Text.RegularExpressions.RegexOptions, System.TimeSpan, Boolean)+142
**System.Text.RegularExpressions.Regex..ctor**(System.String, System.Text.RegularExpressions.RegexOptions)+3e
**Hi.UrlRewrite.Processing.InboundRewriter.TestRuleMatches**(Hi.UrlRewrite.Entities.Rules.InboundRule, System.Uri, System.Text.RegularExpressions.Match ByRef, Boolean)+174
Hi.UrlRewrite.Processing.InboundRewriter.TestAllRuleMatches(Hi.UrlRewrite.Entities.Rules.InboundRule, System.Uri, System.Text.RegularExpressions.Match ByRef)+2b
Hi.UrlRewrite.Processing.InboundRewriter.ProcessRegularExpressionInboundRule(System.Uri, Hi.UrlRewrite.Entities.Rules.InboundRule)+6e
Hi.UrlRewrite.Processing.InboundRewriter.ProcessInboundRule(System.Uri, Hi.UrlRewrite.Entities.Rules.InboundRule)+cd
Hi.UrlRewrite.Processing.InboundRewriter.ProcessRequestUrl(System.Uri, System.Collections.Generic.List`1)+12b
Hi.UrlRewrite.Processing.InboundRewriteProcessor.Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs)+2c4

According to the code you create a Regex instance and it is not added to the cache. All Regex instances that are created manually are not added in code. However, if you execute static Match method instead of instance method, it will create Regex instance and save it to the cache.
So, I recommend you change implementation of the TestRuleMatches method in the following way.
Your code:
    Regex regex = new Regex(pattern, inboundRule.IgnoreCase ? RegexOptions.IgnoreCase : RegexOptions.None);
    inboundRuleMatch = regex.Match(str);
Replace with code:
    inboundRuleMatch = Regex.Match(str, pattern, inboundRule.IgnoreCase ? RegexOptions.IgnoreCase : RegexOptions.None, TimeSpan.FromHours(1));

**Action completed**

We made this change in Hi.UrlRewrite.sln, build and deployed **Hi.UrlRewrite.dll** on all CD servers

**Site outage 3/22/2018 Sitecore response**

I have reviewed the memory dump files.
Indeed GC is triggered and ~250 threads are waiting in every dump until GC is finished.
However GC is triggered on excessive RegExp utilization, which eventually triggered by your custom code.
Please find some call stack examples below:

**\\ Regex should be wisely utilized and may lead to the performance degradation if there are a lot of Rules to Match.**
[[GCFrame]] [[HelperMethodFrame] (System.Threading.Monitor.Enter)] System.Threading.Monitor.Enter(System.Object) System.Text.***RegularExpressions.Regex.CacheCode***(System.String)+6a System.Text.RegularExpressions.Regex..ctor(System.String, System.Text.RegularExpressions.RegexOptions, System.TimeSpan, Boolean)+1f4 System.Text.***RegularExpressions.Regex.Match***(System.String, System.String, System.Text.RegularExpressions.RegexOptions, System.TimeSpan)+41 Hi.UrlRewrite.Processing.InboundRewriter.***TestRuleMatches***(Hi.UrlRewrite.Entities.Rules.InboundRule, System.Uri, System.Text.RegularExpressions.Match ByRef, Boolean)+3b3 Hi.UrlRewrite.Processing.InboundRewriter.***TestAllRuleMatches***(Hi.UrlRewrite.Entities.Rules.InboundRule, System.Uri, System.Text.RegularExpressions.Match ByRef)+59 Hi.UrlRewrite.Processing.InboundRewriter.***ProcessRegularExpressionInboundRule***(System.Uri, Hi.UrlRewrite.Entities.Rules.InboundRule)+b8 Hi.UrlRewrite.Processing.InboundRewriter.***ProcessInboundRule***(System.Uri, Hi.UrlRewrite.Entities.Rules.InboundRule)+1bd Hi.UrlRewrite.Processing.InboundRewriter.***ProcessRequestUrl***(System.Uri, System.Collections.Generic.List`1)+21b Hi.UrlRewrite.Processing.InboundRewriteProcessor.***ProcessUri***(System.Uri, Sitecore.Data.Database, Hi.UrlRewrite.Processing.InboundRewriter)+8b Hi.UrlRewrite.Processing.InboundRewriteProcessor.Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs)+307 DynamicClass.(System.Object, System.Object[])+ae Sitecore.Pipelines.CorePipeline.Run(Sitecore.Pipelines.PipelineArgs)+1d1

**// GetChildren operation should be utilized carefully. In case there are a lot of children, such method would definitely lead to the performance issues. RexExp also introduces additional complexity here. So the call becomes very heavy to execute.**
[[GCFrame]] [[HelperMethodFrame] (System.Threading.Monitor.Enter)] System.Threading.Monitor.Enter(System.Object) System.Text.RegularExpressions.Regex.***LookupCachedAndUpdate***(System.String)+57 System.Text.RegularExpressions.Regex..ctor(System.String, System.Text.RegularExpressions.RegexOptions, System.TimeSpan, Boolean)+140 System.Text.***RegularExpressions.Regex.IsMatch***(System.String, System.String, System.Text.RegularExpressions.RegexOptions, System.TimeSpan)+41 Sitecore.DateUtil.***GetIsoDateParts***(System.String, Boolean, Boolean ByRef)+4e Sitecore.DateUtil.IsoDateToDateTime(System.String, System.DateTime, Boolean)+141 Sitecore.Data.Items.ItemStatistics.GetDateValue(Sitecore.Data.ID)+25 Sitecore.Data.***Comparers***.***CreatedComparer***.***GetCreationDate***(Sitecore.Data.Items.Item)+91 Sitecore.Data.Comparers.CreatedComparer.ExtractKey(Sitecore.Data.Items.Item)+5e System.Linq.Enumerable+***WhereSelectListIterator***`2[[System.\_\_Canon, mscorlib],[System.\_\_Canon, mscorlib]].MoveNext()+6f System.Collections.Generic.List`1[[System.\_\_Canon, mscorlib]]..ctor(System.Collections.Generic.IEnumerable`1)+216 System.Linq.Enumerable.ToList[[System.\_\_Canon, mscorlib]](System.Collections.Generic.IEnumerable`1)+45 Sitecore.Data.Managers.ItemProvider.Sort(Sitecore.Collections.ItemList, Sitecore.Data.Items.Item)+d6 Sitecore.Data.Managers.ItemProvider.GetChildren(Sitecore.Data.Items.Item, Sitecore.Collections.ChildListOptions)+111 Sitecore.Data.Managers.ItemProvider.GetChildren(Sitecore.Data.Items.Item, Sitecore.SecurityModel.SecurityCheck, Sitecore.Collections.ChildListOptions)+3e [[StubHelperFrame]] Sitecore.Data.Managers.PipelineBasedItemProvider.***GetChildren***(Sitecore.Data.Items.Item, Sitecore.SecurityModel.SecurityCheck)+ed Sitecore.Collections.***ChildList.Populate***(Sitecore.Collections.ChildListOptions)+1f Sitecore.Collections.ChildList..ctor(Sitecore.Data.Items.Item, Sitecore.Collections.ChildListOptions)+4b Sitecore.Data.Items.ItemAxes.AddDescendants(Sitecore.Data.Items.Item, System.Collections.ArrayList)+4f Sitecore.Data.Items.ItemAxes.AddDescendants(Sitecore.Data.Items.Item, System.Collections.ArrayList)+12a ***Sitecore.Data.Items.ItemAxes.GetDescendants()***+32 ***CFA.Feature.ListenLoveLearn.Rules.Processors.FormRulesProcessor.ProcessRules***(Sitecore.Data.Items.Item)+86 CFA.Feature.ListenLoveLearn.Factories.RulesFactory`1[[System.\_\_Canon, mscorlib]].Create()+b9 CFA.Feature.ListenLoveLearn.Pipelines.RenderRendering.RenderingMarker.get\_RulesProcessor()+4a CFA.Feature.ListenLoveLearn.Pipelines.RenderRendering.RenderingMarker.get\_RenderDiv()+84 CFA.Feature.ListenLoveLearn.Pipelines.RenderRendering.RenderingMarker.GetStart()+9 Sitecore.Mvc.ExperienceEditor.Presentation.Wrapper..ctor(System.IO.TextWriter, Sitecore.Mvc.ExperienceEditor.Presentation.IMarker)+9d CFA.Feature.ListenLoveLearn.Pipelines.RenderRendering.AddWrapper.Process(Sitecore.Mvc.Pipelines.Response.RenderRendering.RenderRenderingArgs)+11b DynamicClass.(System.Object, System.Object[])+47

I suggest you to carefully review the first two call stacks and the code/logic around the following methods:
**Hi.UrlRewrite.Processing.InboundRewriter.TestAllRuleMatches**
AND
**CFA.Feature.ListenLoveLearn.Rules.Processors.FormRulesProcessor.ProcessRules ( You can ignore this as it’s out custom code )**

I might be possible to decrease the number of rules or introduce the cache layer to optimize the performance.

**Site outage 3/22/2018**

We have completed our analysis on the provided information. Though, we did not notice any hard failure, but based on results, we could attribute this outage to overloaded/saturated server. Based on our analysis here are the findings

* Memory dump file reveals several threads(52, 63, 62 in provided dumps respectively) being locked within Hi.UrlRewrite module.
* **50+ threads from each dump noticed being locked by Hi.UrlRewrite module.** Try disabling it to analyze it's impact on efficiency

Based on assembly analysis we found that couple of the custom classes including **Hi.UrlRewrite uses ItemAxes.GetDescendants() method.**
**ItemAxes.GetDescendants should never be used on large amount of data**, especially if cache sizes were not configured properly. Here is signature of methods we noticed using this method:

< I have removed all other signatures and kept only related to Hi.UrlRewrite >
Hi.UrlRewrite.Processing.RulesEngine.GetOutboundRules() : List<OutboundRule>

**Suggestions:**

1. Consider switching to content search instead of ItemAxes.GetDescendants. Sitecore indexes all items and allows to build comprehensive queries to find necessary items.
2. **Use Database.GetItem method with ID instead of path(if possible). Noticed in 60+ custom methods**